1

AD-A221 717

WELCOME

The Institute for Computer Sciences and Technology and the National Computer Security Center are pleased to welcome you to this Annual Computer Security Conference. The past eight conferences have stimulated the sharing of information and the application of new technology.

This year's conference theme -- Computer Security - Today ... and Tomorrow -- reflects the growth of computer security awareness and a maturation of the technology and its use. The efforts of the National Bureau of Standards, the National Computer Security Center, computer users, and industry have helped to bring about the progress that has been made in the past few years. The commitment of the Federal government and private industry to improve computer security continues to grow, and trusted systems and other technologies are becoming available.

But much more needs to be done. Federal government executive and legislative initiatives for computer security show the extent of national concern. We must strengthen our efforts to make managers, executives, and computer users strong advocates for computer security, and we must make full use of the best affordable technology.

Your participation in this meeting can help to achieve this goal. Let's continue to exchange ideas and then go back to our organizations with renewed purpose and commitment to improve the security of our systems.

[illegible]

JAMES H. BURROWS
Director
Institute for Computer Sciences and Technology

[illegible]

PATRICK R. GALLAGHER, JR.
Director
National Computer Security Center

TABLE OF CONTENTS

Title **Page**

A Brief Summary of a Verification Assessment Study

Richard A. Kemmerer

Department of Computer Science
University of California
Santa Barbara, California 93106

Introduction

This paper is a brief summary of a verification assessment study that was begun in November 1984 and lasted for approximately nine months. The final report (Kem 86), which consists of five volumes, can be obtained from the National Computer Security Center.

The main goal of this effort was a technology interchange among the developers of four established verification systems. The systems investigated were i) Affirm (General Electric Company, Schenectady, New York), ii) FDM (System Development Corporation - A Burroughs Company, Santa Monica, California), iii) Gypsy (the University of Texas at Austin, Austin, Texas), and iv) Enhanced HDM (SRI International, Menlo Park, California). There was some comparative work on examples, but the main idea was for the developers to learn the details of each other's system as a basis for future development.

It was **not** the goal of this study to rate the verification systems that were investigated. It was also **not** the intent of the study to justify the need for formal specification and verification systems or to justify the necessity for research in this area.

The next section gives an overview of the study. This is followed by a summary of some of the issues raised by and conclusions drawn from the study.

Overview of the Verification Assessment Study

The approach taken for this study was first to select a suitable set of example problems to be used to investigate the established systems. Each of the systems in turn was used to specify and verify these problems. The specification and verification was performed by the development team for each system. One member of each system's development team was picked as the "representative" for that particular system. The system representatives were well established with regard to their in-depth knowledge of the particular verification system. In most cases the representative was one of the original developers of the system. The AFFIRM representative was Dave Musser, currently with the Computer Science Branch at the General Electric Company's Corporate Research and Development Center in Schenectady, New York. The FDM representative was Deborah Cooper from System Development Corporation's Santa Monica Research Center in Santa Monica, California. The Gypsy representative was Don Good from the Institute for Computing Science at the University of Texas at Austin in Austin, Texas. The HDM representative was Karl Levitt from SRI International's Computer Science Laboratory in Menlo Park, California. In addition to the system representatives, the assessment team also included two independent participants: Dan Craigen from I.P. Sharp Associates in Ottawa, Canada, and Dick Kemmerer. Tad Taylor, the sponsor's technical liaison, also participated in the process.

At the initial meeting the group agreed on the set of example problems that would be specified and verified using each of the systems. The point of these examples was to determine how the system developers would proceed in solving the problem. It was hoped that ideas as to how these problems should be solved, using the various methodologies, would arise. In addition, it was expected that the strengths and weaknesses of the systems and supporting languages and methodologies would also be uncovered. Finally, it was the hope of the sponsoring agent that these examples would provide insight into how a common set of problems might be used for comparing verification systems.

For each of the four systems, the specification and verification of the example problems was done by the development team for that particular system. The nonresident members of the assessment group then visited the home site of each system to study the system and the solutions to the problems.

During the site visits, each participant was allowed to study the system in any way he or she wished. Usually, this meant that the participant defined a favorite problem and investigated the effects that the system had on the development of a solution. For example, Don Good and Dan Craigen teamed up, for the last three visits, and worked with a micro-modulator example, and Dick Kemmerer worked with a secure terminal example on each of the four systems.

Moreover, the participants concentrated their efforts on areas in which they were particularly interested and tried to understand those parts thoroughly. Dave Musser, for example directed his attention to the theorem proving aspects of the systems. Dan Craigen was interested in language and methodological concerns, and Dick Kemmerer was interested in specifying a large and "real" example.

Through this technical interchange members of both the assessment group and development teams presented their system while the nonresident participants observed the approaches used to specify and verify the example problems.

After visiting a site, each of the nonresident participants prepared a critique of the particular system. After all the site visits had been completed, the assessment team convened at the University of California in Santa Barbara, to compare their findings, to discuss the relevant verification technology issues that were raised during the study, and to propose future directions for verification research.

Technology Interchange

The technology interchange that was the primary goal of this study did occur. One type of technology transfer that occurred was the result of the nonresident participants exercising the systems and discovering bugs and weaknesses for the developers. The value of this type of information is documented in the individual critiques and the responses to the critiques (contained in Volumes II-V of the final report for this study (Kem 86)).

Another type of interchange was from system to system. Several of the developers mentioned during the later visits that they had incorporated (or planned to incorporate) changes to their systems based on what they had learned from the site visits. This was particularly evident during the SRI visit because it was the last visit and because the Enhanced HDM system is in an early stage of development.

The strengths and the weaknesses of the four systems that were observed also served as a basis for formulating the components of a state-of-the-art verification system and for identifying areas that need further research. For example, the move toward a more friendly user interface that was apparent in all of the systems clearly demonstrated the desirability of such interfaces. It also revealed the need to continue to move in this direction incorporating the power of bit mapped graphic displays and windowing capabilities into the verification systems.

Example Problems

One of the conclusions drawn from this study is that the example problems were not "benchmarks". That is, they could not be used to measure the "quality" of a verification system. This result is not surprising, particularly since all of the specification languages are based on first-order predicate calculus, and one can, therefore, specify the same kinds of properties in all of them.

It is also a well known fact that any testing other than exhaustive testing is not complete. The example problems were five test cases that were tried on each of the verification systems. This is not exhaustive testing.

On the positive side it should be noted that the five examples did provide some common ground for comparing and contrasting the four systems. The individual critiques discuss some of the strengths and weaknesses of the systems and languages that were revealed when reviewing the solutions to the example problems.

Formal Verification for Secure Systems

The security community has been a major source of funding for the application of formal technologies and the development of formal verification systems for the last ten years. Their interest is in the use of formal verification to increase their confidence in the security of the systems they are building.

Ever since the Anderson Study defined the reference monitor in 1972 (And 72), security kernels have been an integral part of most secure systems. It is the desire to achieve the third requirement of a reference monitor (it must be small enough to be subjected to analysis and test) that has motivated the security community to embrace formal verification technologies. That is, one form of analysis is formal verification. If one looks up the definition of a security kernel in the Dod Trusted Computer System Evaluation Criteria (commonly referred to as the "Orange Book") (DoD 83) the third requirement has been replaced by "be verifiable as correct". Furthermore, the difference between the highest level of trust (A1) and the next lower level (B3) is "the analysis derived from formal design specification and verification techniques and the resulting high degree of assurance that the TCB (Trusted Computing Base) is correctly implemented."

The reason that the security community turned to formal verification for this added assurance is that testing techniques are not sufficient for giving the desired confidence in the systems being built.

One must keep in mind, however, that that secure systems are just one class of reliable systems (those whose reliability is defined in terms of security requirements). Therefore, the benefits of formal verification are the same as for any reliable software development project. The use of formal verification techniques helps to avoid sloppy thinking and the verification systems keep one "honest". That is, by using formal specifications one can precisely document the requirements of a system in unambiguous terms. Furthermore, because the specifications are written in a formal notation, one can reason about the specifications and one can also analyze them using computerized tools. Thus, properties can be proved about the specifications.

In summary, the security community's motivation for using formal verfication techniques is no different than those of anyone wanting reliable software. There is a difference, however, in the properties proved.

Formal Semantics and Mathematical Justification

One of the issues that was raised during the study was the need for formal sematics for the specification and programming languages and a mathematical justification for the proof approach being used. There was a consensus within the group that formal semantics and a mathematical justification would be good to have. However, there was a difference of opinion about the role of these mathematical foundations in verification system development. The question raised by the assessment group was "is it necessary to have the formal semantics and the mathematical justification all rigorously defined before building a system or is it better to begin building a system while having only a partial formulation of its foundations?" This issue was not resolved during the study. Some participants felt that it is necessary to have the formal semantics play an active role during the design of the specification languages, programming languages, and the underlying logic. Therefore, the formal semantics should be fully defined before going off to build a verification system. Other team members felt that if one insisted on formal semantics before anything else, the verification system might never get built, and that one can have useful systems without fully defining the formal semantics. The four verification systems that were investigated in this study were built without the formal semantics being fully defined (if defined at all). However, Don Good remarked that he thought that not having developed a formal semantics for Gypsy as a part of the language development was one of the most serious mistakes made in the Gypsy effort.

Design Verification

Another issue that was raised during the study is "what is design verification and of what use is it." The DoD Trusted Computer System Evaluation Criteria requires design verification for systems rated at the highest level of confidence (Division A systems). The orange book (DoD 83) defines design verification as

> "the process of using formal proofs to demonstrate the consistency between a formal specification and a formal security policy".

An identical definition is given in the COMPUSECese Computer Security Glossary (DoD 85).

After much discussion the issue was reduced to whether "design" verification is the appropriate term. What is being verified is a specification and not a design, although the specification may be part of the design process. The group was also concerned that the term "design" carries a connotation of being complete while a specification is often incomplete.

To help settle the question the available software engineering texts (approximatly ten of them) were consulted to determine the appropriate definition of "design". The hypothesis was that design was a well understood term in the software engineering community. The surprising result of this search was that most of the definitions of design were ambiguous, and of those that were not ambiguous, there was little agreement as to what constitutes a design. Because the investigation revealed that the term design was not as well understood as was originally thought, the group decided to take a fresh look at the process that was being defined to determine if there was a more accurate term for the process

The consensus was that a specification was a description of some property(ies) of a system. Furthermore, security models are high level specifications. The group also agreed that it was useful to prove properties about specifications, and that testing and proving properties about specifications (possibly even incomplete specifications) is one way of gaining confidence that the specifications satisfy some desired properties.

The conclusion was that what was taking place were proofs about specifications. Therefore, the term "specification verification" more accurately describes the process commonly referred to as "design verification".

Is Formal Verification a Stagnant Field?

It has been suggested, especially during the last verification workshop, that the field of formal verification is in a state of stagnancy. The particular observation made at VERkshop III (Ver 85) was that very little seems to have been accomplished since the previous workshop (held four years earlier).

Computing science in general, and formal verification in particular, are addressing some very real and difficult problems. Formal verification is a multidisciplinary field. It requires understanding of programming and specification languages, programming and specification methodologies, mathematics (both for reasoning about programs and and systems and for describing languages), and system interfaces. To engineer these technologies into a cohesive whole is extremely difficult, but the payoffs could be substantial. If the development of verification systems has been slow, it is because of these fundamental challenges.

A Production Quality Verification System

The four verification systems examined in this study represent the leading edge of mechanical verification technology. This mechanical support is useful, if not necessary, when applying formal verification to real applications. However, each of these systems has been built primarily as a research vehicle for exploring different ways of implementing and applying formal verification. None of them has been designed or implemented as the kind of production quality system that is needed to support wide-spread application of verification to real software systems. There is much that needs to be done to progress from where the systems are now to a truly production quality verification system.

The most important requirement for a production quality verification system is soundness. Soundness for a verification system means that if the verification system claims that an application is proved and the assumptions underlying the verification system are true (correct hardware, compiler, etc). then the application actually will exhibit the properties that the verification system claims to have proved about it. Without soundness, the results of a verification (be it mechanical or otherwise) cannot be trusted. If a verification system is to be used in any important application, soundness must be given top priority.

Each of the research prototypes that were studied has some areas of unsoundness. Often the cause of this unsoundness simply is that the implementations of the existing systems are incomplete in some important way. These problem areas usually can be avoided or finessed by an expert user; but this level of expertise cannot be assumed for the potential user community of a production system. As mentioned above, one way in which all of the four systems are incomplete is that none of them have a fully developed, mathematically precise definition of the semantics of the languages they process. This is the standard against which a rigorous determination of the soundness must be made.

A production quality verification system must be well engineered. It needs to have a high quality user interface. It must perform efficiently. It must be robust, well documented, maintainable, etc. It should be built with the best methods available for software engineering, quality control, configuration management, etc. Generally, these issues have not been given a high priority in the implementation of the research prototypes, and all of them have major deficiencies in some of these areas. The current systems have been developed primarily to demonstrate the feasibility of i) mechanizing formal verification and ii) applying it to real software systems. They have served that purpose well, but they are far from being production quality systems.

A production quality system that is to be used by a large community must be hosted on equipment that is readily available to that community. The National Computer Security Center has taken a first step in this direction by making the FDM, Gypsy, and HDM (both the original and the enhanced) verification systems and the Boyer-Moore theorem prover available a Multics system on the ARPAnet. This is a reasonable first step; however, due to the limited Multics user community (as compared to TOPS20, or UNIX) and the dissatisfaction of having to work over the ARPAnet, some other means of reaching a wider audience must be found.

If a production quality system is to be made available on a wide-spread basis, education of the potential user community will also be required. This community will need to be educated in the fundamentals of verification as well in the use of mechanical verification tools. The existing research prototype systems can play a useful role here. They can be used to help educate the community, and they can be used to explore a wider variety of applications of formal verification. Demonstrating the effectiveness of verification on an increasing variety of important applications probably is the best way of drawing the attention of the software engineering community to verification, and thereby accelerating its development.

Verifiability of Ada

Currently, there is a significant degree of interest in determining whether the programming language of Ada is amenable to formal program verification techniques. This interest is particularly evident in the security community. This interest in Ada Verification is most likely the result of the following line of reasoning. DoD Directive 5000.31 states that Ada is to be used for all mission critical embedded systems software. It is reasonable to assume that secure systems are mission critical. Furthermore, secure systems that are to be certified at the A1 level require formal verification. Therefore, it is reasonable to assume that Ada verification may be required for secure systems.

This line of reasoning may seem plausible; however, it should be noted that no DoD requirement for code verification exists. At the A1 level the requirement is for a manual or other mapping between the formal specification and the source code, to provide evidence of correct implementation.

While it was the collective opinion of the assessment group that a verifiable subset of Ada can be found, the group also believed that it was necessary to note some important observations and concerns.

The main problem noted is that Ada is a particularly complex language. As a result, finding a useful and easily determinable axiomatizable subset of Ada is a difficult task. The group concluded that before building an Ada verification system time should be spent trying to understand the components of Ada that contribute to its complexity.

Research Directions

It was concluded that what is needed to make a significant advance in the use of formal verification for reliable software is a variety of "exploratory applications" that explore the potential utility of verification technology. This experimentation shuld result in a variety of publicly visible examples that show the benefits of formal verification. It would also be desirable to have a technology that gets accepted without being mandated by the National Computer Security Center or the government in general. That is, one would like the general public to view the examples and conclude that this is how they would like to build their systems.

To achieve success would require experimentation on a wide variety of examples. It would be beneficial to have the academic, industrial, and government communities all involved in this experimentation. To carry out the examples would require a long term commitment from funding agencies.

One of the side effects of these experiments is that the limits of the verification techniques would be made known and the areas in need of further research would be exposed. Another benefit of the experiments would be production quality systems, for without them there would be no hope of public acceptance.

Conclusions

It should be noted that the conclusions contained in this paper are the result of looking at four verification systems. Although the assessment team members brought a large amount of formal verification knowledge to the study, the reader should be aware that this is a view of the total field of formal verification. It was evident from this study that although it is possible to gain valuable insight and understanding during a one week visit, it is impossible to fully understand a system in such a short time.

Acknowledgements

I would like to thank the members of the assessment team for their dedication to this study. They approached the study with open minds that provided a refreshing academic atmosphere for exchanging ideas and knowledge.

I would also like to thank the associations that hosted each of the site visits for giving so generously of their time and facilities. Particular thanks goes to the development teams for each of the systems who made each of the site visits an enjoyable learning experience.

References

(And 72)	Anderson, J.P., Computer Security Technology Planning Study, ESD-TR-73-51, Vol. I, AD-758 206, ESD/AFSC, Hanscom AFB, Bedford, Massachusetts, October 1972
(DoD 83)	Department of Defense Trusted Computer System Evaluation Criteria, CSC-STD-001-83, Department of Defense Computer Security Center, Fort George Meade, Maryland, August 1983
(DoD 85)	COMPUSECese Computer Security Glossary, NCSC-WA-001-85, National Computer Security Center, Fort George Meade, Maryland, October 1985
(Kem 86)	Kemmerer, R.A., Verification Assessment Study Final Report, Volumes I - V, C3-CR01-86, National Computer Security Center, Fort George Meade, Maryland, January 1986
(Ver 85)	Proceedings of VERkshop III -- A Formal Verification Workshop, Pajaro Dunes Conference Center, Watsonville, California, February 1985, Software Engineering Notes, Vol. 10, No. 4, August 1985

A NETWORK SECURITY PERSPECTIVE

Jonathan K. Millen
The MITRE Corporation
Bedford, MA 01730

1. INTRODUCTION

BACKGROUND

Network security is roughly at the same stage ADP system security was about ten years ago, when prototypes of the first multilevel secure systems were being built. Systems with some degree of security already existed, but it was important to have systems that were more flexible (including the ability to support DoD needs), and which provided an assurance of security based on more than a limited amount of testing and a firm handshake.

Secure networks, in some sense, are all around us. The ARPANET has been used to carry classified information, using PLI's (BBN's Private Line Interface) to provide end-to-end encryption. Circuit-switched networks employ link encryption devices to set up secure channels. Banks use DES-based encryption to protect funds transfers. But modern packet-switching networks present many opportunities and problems that have not yet been fully explored.

The phased development and growth of DDN as a backbone network to carry classified information, and the development of distributed application-level networks such as SACDIN and DoDIIS that will make use of its services, make it necessary to understand and plan for the more advanced capabilities envisioned for the future, as well as the concerns arising from the interconnection of a wide variety of ADP systems in a common internet environment.

Many important network security issues were brought out in a Spring, 1985 DoD Workshop organized by the National Computer Security Center (NCSC) [1]. The objective of the Workshop was to provide the NCSC with input for the development of trusted network evaluation criteria, analogous to the Trusted Computer System Evaluation Criteria (TCSEC) [2]. The network criteria would provide technical guidance for the DoD in the evaluation and acquisition of networks in which security needs are significant. It was clear that much of the organization and content of the TCSEC applied to network evaluation, but also that the TCSEC was deficient or inapplicable in some respects for this purpose. The TCSEC needed to be revised, replaced, extended, or at least reinterpreted for network evaluation.

A number of issues discussed in the Workshop are still unresolved. Some of the unresolved issues are very basic, such as whether the current state of the art is adequate to certify any networks as secure at an "A" level, implying a high assurance of security comparable to A-level standalone systems. Another basic concern is the scope of the criteria; what exactly is a network, and what kinds of networks can reasonably be evaluated?

ISSUES OVERVIEW

The need for certain significant additions and changes in the TCSEC to adapt it for network evaluation emerged from the Workshop. Some of the more notable characteristics of network evaluation that distinguish it from the standalone system evalution are summarized below. While each of these characteristics is a response to issues that demanded attention, there are, in some cases, disagreements about how to deal with them; those disagreements are indicated below as well.

The characteristics touched upon in this subsection are: the global vs. component view of a network, trusted paths, interconnection rules, communications integrity and denial of service, treatment of non-host components, and encryption. The following subsection begins the main topic of this report, the relation between security policy and protocol layering, and how it should affect network evaluation.

One pervasive theme of the Workshop was the need to view a network both as a global entity with a single security policy, and as a collection of components that must be individually specified and evaluated. For secure operation, a network is bound to have certain standards, restrictions, and conventions that must be obeyed and enforced network-wide to obtain assurances that all users can count on. In particular, some networks will have centralized facilities like access control centers and repositories of audit information whose proper use must be specified and enforced globally.

At the same time, networks are normally developed by connecting together a variety of different components with different functions, many of which existed independently prior to the network or were off-the-shelf commercial products. It is important to foster the development of future products of this sort by understanding how to evaluate their designs on their own, to the extent possible, out of the context of any specific network.

The trusted path requirement is an extension of an authentication requirement found in secure operating systems. In a standalone system, there are times when a human user must communicate directly with trusted software, without any possibility of undetected interference or forgery by untrusted software. This occurs, for example, when a user is presenting a password for login, since it should be read only by trusted software; and when a privileged operation is being requested, since the request should be honored only from an authorized person. In a network, there are times when trusted processes in different sites must share a similarly protected channel. For example, one host may relay a local user's password to a remote host, or send security-related reconfiguration instructions from a local network administrator to a remote site.

The interconnection rules are a statement of mandatory access control policy at a level of abstraction (or a layer of protocol) for which the network provides data links between potentially multilevel components. These rules are an explicit assurance to host administrators that their data will not be sent to other hosts that are not accredited to receive it. The current rules assume that data links are bidirectional, because of the usual need for acknowledgements and other two-way coordination when setting up connections. In some applications there is a need for true one-way data flow, and there is a question whether one-way data links should be recognized by the interconnection rules, or whether they should be treated as something that occurs invisibly inside a trusted component.

The requirements relating to communications integrity and denial of service result from the general feeling that these concerns, while already present for standalone computer systems, are more serious in a network context, because of the greater vulnerability of communications links to random errors, wiretapping, and other threats affecting data in transit. Hence, though they are not mentioned explicitly in the TCSEC, some security requirements of these types should be imposed on networks. However, the Workshop results indicated that the definition of "denial of service" is mission dependent, and hence it would be difficult to define general requirements for countermeasures against it. Similarly, while communications integrity can be quantified statistically, it is difficult to state universally acceptable requirements for transmission accuracy.

The idea of trying to apply the TCSEC to network components seems to work well when the component is a multilevel host, but is less plausible when the component has a more limited or special function, such as an encryption device or a switching node. Even hosts, multilevel or otherwise, are not evaluated in the same way for network purposes when the network connection is limited to a single security level, or when the hosts has special trusted functions introduced to support the network connection.

Encryption plays an important part in network security, but it is not clear to what extent requirements for particular encryption methods, and for the associated software and hardware, can be specified in a document analogous to the TCSEC. The reason for this is the division of responsibility between the NCSC, which is competent to evaluate trusted software, and those parts of NSA and other organizations that are competent to evaluate cryptosystems.

LAYERING SECURITY POLICY

There are a number of terms and concepts in the TCSEC that are difficult to interpret in a network context. Two of the more troublesome ones are *subject* and *object* . Even for standalone computer system evaluation, it is not always clear what the subjects and objects of a system are. Subjects are usually human users or processes; but sometimes I/O ports can be regarded as subjects. Objects are usually files; but sometimes I/O devices, temporary internal buffers, and subjects are considered to be objects.

In practice, subjects and objects are identified in the context of a particular system in conjunction with the access policy it is designed to support. In other words, the interpretation is a judgment call. If it turns out that certain repositories of information have been neglected as candidates for being objects, transfers of information through them will be regarded as covert channels. Since covert channel analysis is part of TCSEC evaluation, the situation is reasonably under control.

The situation is more complex in a network environment. There are many more options for the interpretation of subjects and objects. Hosts, nodes, gateways, switches, front end processors, and subnets might also be subjects: and messages, packets, virtual circuits, connections, channels, links, headers, plus the new subjects just named, might also be objects. The prospect of devising an access control policy for an internet that delineates the roles of many of these players, and performing a covert channel analysis that takes care of the ones that were left out, is daunting.

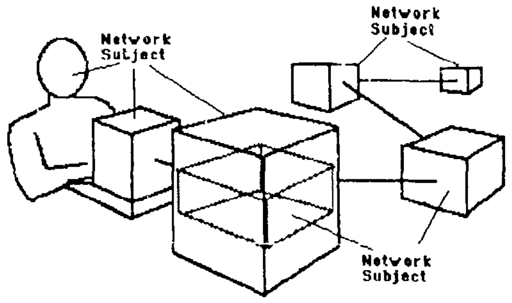


Figure 1 Are They All Network Subjects?

A problem related to the interpretation of subjects was discussed in the Accountability group at the Workshop, and its conclusion hinged on the concept of protocol layering. The group noted that individual identification and accountability across a network is a service provided by a high layer of protocol. Individual accountability is not possible in networks that only provide services up to the transport layer. A transport service is host-to-host; it has no way of knowing whether a particular user is receiving data from a particular file.

The point is that *whether a security policy makes sense depends on the service provided by the network, as specified by the user interface to a particular protocol layer.* The binding of a security policy requirement to a protocol layer is quite natural in a network context, and it should provide some insight, not only into how to interpret policy requirements, but also how to structure the evaluation process.

It is not necessary to confine the security policy for a given network to a single protocol layer interface. Requirements on different layers will certainly be called for. For example, all information on communication links should be protected from undetected eavesdropping or other interference, by physical protection or link encryption. That requirement clearly applies to the logical link layer or below, and exists independently of higher level requirements on access control.

Stratifying security requirements into protocol layers has the principal benefit of preventing interpretations that are nonsensical, or fundamentally incompatible with the way networks are designed, because they cross layers. It also has implications for
how to define what a network is, for purposes of evaluation, and how to identify and evaluate its components.

To quote Tanenbaum, "The peer process abstraction is crucial to all network design" [3, p. 13]. Peer processes communicate with one another following certain rules defining message types, formats, and conventions for various activities such as opening and closing connections, error correction, and so on. In order to send a message to a peer, a process uses a lower protocol layer, sending the message downward through an interface; and the other process will receive it when it pops up through the interface at the destination. Although the two communicating processes may be a considerable distance from one another, the interface to the lower protocol layer forms a single conceptually global shell, enclosing a system that is itself a network.

We might visualize peer processes as heads of pins, which are all stuck in the same pincushion. The pincushion is hollow, however, and inside there is another pincushion complete with pins, whose heads form the shell of the outer one. Similarly, the peer processes in the higher layer may support a higher level network service. We are, therefore, equating a network with the service provided by a protocol layer, and observing that networks can be nested within others by virtue of the protocol layering.

This layering of networks is not merely an abstraction; network services are actually built by adding components supporting a higher level protocol to an existing network. This sort of network construction suggests that the entire existing network should be thought of as a single component of the new, higher-level one, since it was one of the "pieces" used to put it together.

THE ISO MODEL

We will make use of much of the ISO reference model terminology because of its wide familiarity. In that model, the architecture of a network has seven layers, and those layers will be characterized briefly below. It should be kept in mind that the existence of layers, and the occurrence of certain common functions, are more important than the particular grouping of functions into the ISO layers. Few, if any, networks have natural separations between layers at the exactly the same places envisioned by the ISO committee, and many networks have additional functions that do not seem to belong inside any of the seven layers, but occupy layers of their own. Nevertheless, the seven ISO layers are helpful as a starting point.

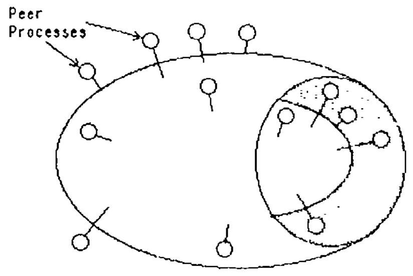


Figure 2 Protocol Layering

In layer 1, the physical layer, the peer entities are simple transmission and reception devices such as modems. For each of them, the network is only a single wire leading to another modem. A modem is also conscious of a user who communicates with it over a connector, acting as a input and output for voltage levels. Voltage levels at some of the pins on the connector are simple commands to start, wait, etc. Security concerns at the physical layer are limited to physical protection of the link medium from tapping or electromagnetic eavesdropping.

In layer 2, the data link layer, the peer entities are processes who see the network as a kind of two-ended modem (a modemedom?) that can be used to transmit individual 8-bit characters, or perhaps longer data units, to a corresponding process at the other end of the modem. These data link processes may be located in a host or in a separate network interface unit. Their users are sources and sinks of character streams. A data link process may have some responsibilities for error detection and retransmission. Link encryption is typically applied at the lower edge of the data link layer.

In layer 3, the network layer, the peer entities are processes who see the network as a collection of communicating processes - this is the first layer that knows that a network has more than two ends. Let us refer to each of these processes as a "node". A node understands that it is connected directly via data links to only a few other, neighboring, nodes, and often plays the role of a relay station, passing on packets received from other nodes. Its user, if any, is a host process. A packet is a sequence of characters with source and destination addresses, plus some error detection information relating to the packet as a whole. Some communications integrity concerns are addressed at the network layer.

In layer 4, the transport layer, the peer entities are processes representing hosts. A host process is actually less aware than the layer-3 nodes of the network topology; it knows the addresses of other hosts, but it doesn't know which ones, if any, are its neighbors. The host process divides its user input into packets. If necessary, it attaches a sequence number to the packets, so that its peer entity will know when packets have arrived out of order, and thus can reorder them, and detect when packets are missing. End-to-end encryption can be applied at the bottom of the normal transport layer.

In layer 5, the session layer, the peer entities are processes whose users are application programs. An important function of a session-layer process is to set up a connection by going through a login procedure, which may involve communication with a peer entity in an access control center host. When end-to-end encryption is used with automatic key distribution, a session-layer process uses transport layer services to obtain and distribute the encryption key.

The two higher layers, the presentation layer (6) and application layer (7), differ greatly from system to system. In a distributed system, where the user is not forced to distinguish between local resources and remote resources, processes at these layers translate user requests that require remote resources into requests for session layer services. Most network security concerns are addressed at lower layers, though end-to-end encryption could, in principle, be applied in any layer from 4 to 7.

In an internet environment, host addresses accepted by the transport layer have a network component, so that hosts in other networks may be addressed. Internet communication is accomplished by forwarding packets from one network to another via gateway hosts. A protocol layer is needed to translate the compound net/host addresses into an appropriate host or gateway address within a network. The internet layer is also concerned with fragmenting and reassembling packets at gateways for travel through networks with different packet sizes. Since the internet layer is used by the transport layer and, in turn, uses the network layer, it is between layers 3 and 4, as described above, and it is viewed as the upper part of layer 3.

2. A SEQUENCE OF EXAMPLES

INTRODUCTION

The importance of protocol layering in evaluating networks will be illustrated with a sequence of examples based loosely on the evolving DDN architecture. We will look at several networks, each one built on top of a preceding one. In each case we will perform an off-the-cuff evaluation of the network under a reasonable interpretation of the TCSEC, with respect to compromise protection. The examples are intended to bear a general architectural resemblance to certain real networks, such as the ARPANET. In some cases, the names of the corresponding real networks will be used for the examples to suggest the connection, but it should also be kept in mind that many details and features of the real networks have been omitted or altered.

Security policy requirements will be applied to the network service provided by the outermost protocol layer, while architectural requirements will be applied, where appropriate, to network components.

ARPANET

We begin with a simplified version of the ARPANET. The basic components of this ARPANET are the IMPs (Interface Message Processors, which are switching nodes) and the trunks, providing a network-level host-to-host service. The network provides discretionary access control, as required for division C, in the sense that messages are delivered normally only to the addressed destinations. This seems to satisfy the requirement for access control at the granularity of a single host.

The discretionary access control requirement actually refers to "users", but the network provides only host-to-host service, so the only proper interpretation for "user" here is "host". Identification and authentication in the usual sense are obviated with this interpretation for "user".

Looking at the architectural requirements for class C1, one could say that the TCB (Trusted Computing Base) operates in its own "domain", since we could include all the software in each IMP in the TCB; there is no "user programming" on this system.

Yet this ARPANET has a serious security problem: any individual could obtain information destined for any host by eavesdropping, via wiretaps on suitable trunk lines. There is, of course, no reference to this kind of vulnerability in the TCSEC. Should we disqualify this network for division C, or just say that it is good enough for C1 but not for C2? One way to pursue this question is to look at a similar network that addresses this vulnerability.

PRIVATENET

Suppose that link encryption devices are added to trunks between IMPs, and at the same time we place the IMPs into secure areas. The net effect of these measures is to protect sensitive information from exposure to the outside world. Although the host interface to the network is the same, its link-layer service component has been replaced with a more secure one. This makes it a new network; call it PrivateNET.

The most startling difference between ARPANET and PrivateNET is that the latter could operate in a dedicated or system-high mode with classified information, (assuming that the link encryption system was approved) while the former could not, unless it were a local-area network entirely enclosed in a protected facility. It is true that any standalone computer could process classified information if it were enclosed in a protected environment, without raising its evaluation class. Nevertheless, it is argued here that encryption should be regarded as an architectural feature of the network and not an environmental add-on, because it changes the nature of the service offered to users. This is perhaps not so compelling in the case of link encryption, since the associated encryption devices are relatively simple. In more advanced schemes, however, in which access control is interwoven with key distribution, it is clear that the architecture of the encryption system is a large and significant part of the network design, with substantial trusted software, and it must receive correspondingly great attention during the evaluation.

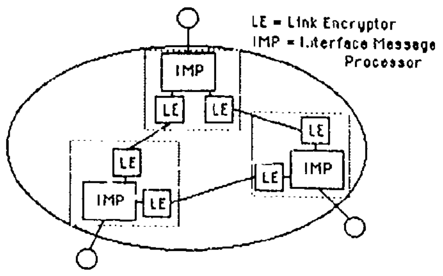


Figure 3. PrivateNET

Is ARPANET or PrivateNET in class C2? Possibly. The requirement for "resource isolation" suggests that special provisions are necessary to prevent messages from getting mixed up inside switching nodes. It is unclear whether the IMPs satisfy this requirement. However, the software that keeps messages separate is no worse than the software that supports the discretionary access control requirement by preventing misdelivery, so there does not appear to be a reason to reject it. Another factor to consider is the maintenance of an adequate audit trail.

IPLI-DDN

On top of the PrivateNET basic transport service we can superimpose a layer that provides end-to-end encryption, initially with IPLI's (Internet Private Line Interfaces). This is a new network, also with an interface to a layer 3 service. To ensure that a message will be kept secret from all hosts other than the desired destination, one arranges (ahead of time) for the IPLI's at one's own host and the destination host to share a key that is not available at any other. Or, one could arrange for group-level access control by distributing keys on a community basis. This scheme is very much like one of the architectures suggested for a pre-Blacker phase of DDN, although subsequently discarded. Let us call it IPLI-DDN.

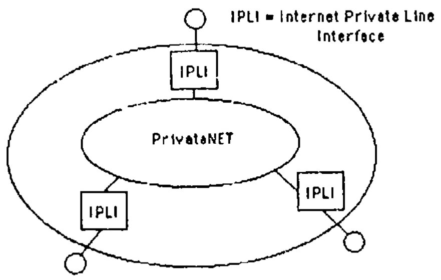


Figure 4. IPLI-DDN

End-to-end encryption gives us much greater assurance that messages will not be compromised by either eavesdropping or misdelivery. But the network is still only in division C, because it knows nothing about security level labels. Given the large amount of additional expense and effort that went into it, relative to PrivateNET, and its greater level of protection, it deserves a higher ranking.

With IPLI-DDN we have network complex enough so that we need to take a close look at its components. What are the components of IPLI-DDN? The IPLI's are certainly components; and it is suggested that the entire PrivateNET be taken as the only other component. The TCSEC has security policy, accountability, assurance, and documentation requirements for a TCB that have implications for each component. These requirements could reasonably be supported by an IPLI, though some effort and perhaps some new documentation would be necessary to establish that claim. Some of the requirements, especially those relating to accountability, apply rather obliquely when a host is a network subject.

The PrivateNET component needs to be trusted only to support discretionary distinctions between hosts in the same key community. But this property may be inferred from its prior "evaluation" as a network in its own right. This illustrates how certain short cuts are possible when a subnet can be regarded as a single component of a higher-level network.

DNSIX

As an example of a network supporting mandatory access control, consider multilevel security facilities such as those planned for DoDIIS (DoD Intelligence Information System). Let us assume, for our purposes, that DoDIIS will depend on IPLI-DDN for backbone communication over long distances. A DoDIIS node consists of one or more hosts with a common interface to IPLI-DDN. DoDIIS hosts generally handle compartmented information, but only some operate in true compartmented mode, while others are system high, and still others are dedicated to a single compartment.

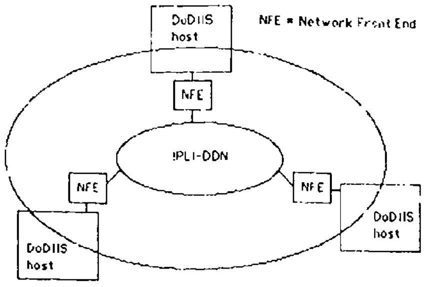


Figure 5 DNSIX

The network security architecture being developed for DoDIIS is intended to support controlled access by users at terminals to FTP and Telnet services at remote hosts. The security policy has implications for (1) restrictions on creating cross-network sessions and (2) security labels on datagrams. The policy is to be enforced by a additional protocol layer (or layers) called DNSIX (DoDIIS Network Security for Information eXchange). The DNSIX software is split between each DoDIIS host and its associated NFE (Network Front End), which contains the TCP/IP software.

When evaluating DoDIIS against division B requirements, the network service we are actually evaluating is the DNSIX interface, which provides the remote services. The requirements for DNSIX do appear to match closely with B requirements.

The components of DNSIX are (1) the DoDIIS hosts, since they have trusted DNSIX software; (2) the NFE's, since they also have trusted DNSIX software; and (3) IPLI-DDN. Considered as components, each of these has certain specific functions it must support, and its evaluation is with respect to these functions rather than the overall mandatory security policy which they support. While the compartmented DoDIIS hosts will probably be B-division systems with respect to the TCSEC, that fact is not relevant to their evaluation as DNSIX components, except insofar as their architecture assures the protection of the DNSIX software they contain. Similarly, even though IPLI-DDN is only division C, it can be a component of an B network, because its function is only to isolate connections: the mandatory access policy is taken care of by the DNSIX protocol layer.

It should be kept in mind that installing DNSIX software in a DoDIIS host may have repercussions on the TCSEC rating of that host. For example, the DNSIX host software may have privileges such as kernel-domain access to communications ports. If so, it is trusted not only in the network sense, but also for the host evaluation. Recertification of the host may be needed.

It is also reasonable to try to evaluate DoDIIS itself, rather than its network interface DNSIX. DoDIIS can be regarded as a distributed system, providing access to both local and remote services. The interface to the trusted part of the system, which should be identical to the TCB interface in each host, is very much like a protocol layer. Distributed system evaluation is discussed further in the next section.

BLACKER-DDN

Another major step in upgrading DDN is to use Blacker for end-to-end encryption instead of IPLI's. Like IPLI-DDN, a Blacker-DDN is built by putting a protocol layer on top of PrivateNET. Blacker-DDN components include not only the Blacker Front End (BFE) in place of the IPLI, but also a Key Distribution Center (KDC), and an Access Control Center (ACC). The subnet component is PrivateNET. The functional advantage of Blacker over IPLI's is that keys are distributed in such a way as to enforce security level separation as well as community separation. It is also more convenient because keys are distributed automatically over the network.

Because Blacker-DDN enforces interconnection rules based on security levels, it should be targeted for division B or A. In the TCSEC, the step from B to A comes primarily from the use of more rigorous methods to verify that the compromise protection policy is upheld.

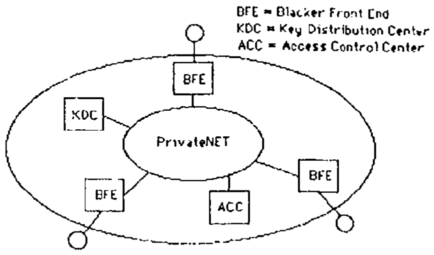


Figure 6 Blacker-DDN

Suppose for a moment that the additional verification effort were not made to raise Blacker-DDN from B to A. We would then have two networks, Blacker-DDN and DNSIX, both in B, but with significant architectural differences between them. Although Blacker-DDN and DNSIX both support a mandatory access control policy, the special Blacker components will be designed with more attention to the separation of security-critical modules from the rest of the system. Another way of summarizing the difference is to say that Blacker components can be evaluated under the TCSEC as B3 or A1 systems, while the DoDIIS hosts and NFE's are probably only B2 at most. This means that there are environments (or distributed systems) for which Blacker would be satisfactory and DNSIX unsatisfactory. This suggests that it is reasonable to maintain the distinction between B2 and B3 in a network context on the basis of architectural requirements, so that Blacker-DDN could be distinguished from DNSIX.

AUTO-DDN

There is an alternative to using end-to-end encryption. We could, instead, upgrade PrivateNET by replacing the IMPs by special packet switching nodes (PSN's) that inspire greater confidence, by virtue of their architecture and development environment. They might, for example, contain security kernels and be evaluated under the TCSEC at a relatively high level, perhaps even A1. Let us call this hypothetical network AUTO-DDN; it is reminiscent of AUTODIN II, whose development was discontinued in favor of DDN.

AUTO-DDN is not built on top of either PrivateNET or ARPANET. Like PrivateNET, it is built on an encrypted link layer. The components of AUTO-DDN are the PSN's and the link layer. If it were evaluated, its rating would depend on the functionality and architecture of the PSN's. Let us suppose that the PSN's support mandatory access controls, so that, say, a Secret datagram will be delivered only to a host accredited for Secret information.

If we compare AUTO-DDN to Blacker-DDN, they are similar in the quality of their components, but there is a striking difference in the protection of message data in switching nodes: it is protected by end-to-end encryption in Blacker-DDN IMPs, while it is in the clear and protected only by the operating system access controls in AUTO-DDN PSN's. This is certainly a large enough increment in compromise protection to warrant evaluating Blacker-DDN at a higher rating.

Comparing AUTO-DDN with DNSIX is more difficult; we seem to be comparing apples and oranges. Since DNSIX is built on IPLI-DDN, it provides end-to-end encryption of message data in IMPs; but AUTO-DDN PSN's employ a more trustworthy architecture (by assumption) than the DoDIIS hosts and NFE's with their DNSIX software. The crucial observation here is that the data is still in the clear while in the DoDIIS hosts and NFE's; the IPLI's provide only community isolation. Consequently, the risk of mislabelling message data is greater in DNSIX. This argument supports the contention that DNSIX, AUTO-DDN, and Blacker-DDN (before verification) exemplify three classes within division B.

3. DISTRIBUTED SYSTEMS

INTRODUCTION

One of the conundrums discussed at the Workshop was whether to think of a network as simply a communications service joining independent hosts, or as a distributed system into which hosts and communications are integrated.

The term "distributed system" is normally reserved for a network that offers application-layer services, and controls access to both local and remote resources. DoDIIS and SACDIN are examples of distributed systems. At the smaller end of the scale, there are distributed systems on local-area networks (LANs). There are many examples of workstations on a LAN sharing a global file system, in which a file located at another workstation or a file server can be loaded as easily as one stored locally.

The term "distributed system" can also be used in a broader sense to apply to any network, inclusive of the hosts that are connected to it. It is convenient for us to use the term in this broader sense, since we have restricted "network" to mean a protocol layer interface. In this section we will look at concerns that arise from the way hosts are connected to networks to form distributed systems.

In a "true" distributed system, network access to remote resources is viewed as an extension of the local resources provided by each host. It was stressed earlier that a global network security policy should be stated in the context of the service interface to one or more protocol layers, so that the appropriate subjects, objects, and access control requirements can be identified. When thinking in terms of a distributed system that manages both local and remote resources, we should still be able to identify a *distributed service interface* in terms of which to state the policy, even though it is not strictly a protocol layer interface.

For true distributed systems it is reasonable to follow our general prescription for applying the TCSEC to networks: apply security policy requirements to the global interface, and architectural requirements to the components, including, in this case, the hosts. But the implementation of this approach will not be smooth sailing. The principal difficulty will be in deriving its implications for non-host components.

COMPONENTS

It will be necessary to limit security policy requirements of non-host components to match their specific functions. The design specification and verification requirements for division B and A components can be seen as limited to security properties needed to support a global policy. This means excluding TCSEC requirements for security labels that may be inappropriate for some trusted components. A switching node, for example, must be trusted to separate messages from one another, and prevent message data from leaking into headers; but it can do so with no need to maintain security labels.

The perspective espoused in this paper suggests that it would be very desirable to view subnets as components; the problem is that TCSEC architectural requirements are really applicable only to standalone computers. As an expedient one might say that, when a distributed system is built on top of a subnet, like PrivateNET or IPLI-DDN, all of the components of the subnet (and all of their components, etc.)

are elevated to the status of components of the distributed system, down to every IMP, gateway, and modem; but it would be conceptually simpler, and more in tune with the precepts of network architecture, if that were not necessary.

The above considerations suggest that special requirements should be developed for various specific types of components, including subnets. Security policy-related requirements and architectural requirements would both be tailored for the type of component.

Having separate requirements for different kinds of components could also facilitate a more detailed consideration of the security features appropriate for them. It might become practical to implement the recommendation of the Components Group at the Workshop, namely, to rate different features at different assurance levels. A link-encrypted wire, for example, as a subnet component, provides host-granularity discretionary security (a C-division feature) with an extremely high (A-division) assurance.

COMPLEX SYSTEMS

A host attached to a network has schizophrenic roles as a provider of both local services and network services. True distributed systems integrate hosts coherently into the network, but in others the network connection is an afterthought. In the latter case, it may be impractical to identify a distributed system service interface that supports a coherent security policy.

[illegible] stems like SACDIN and I-S/A AMPE, whose hosts [illegible] architecture and evaluation rating, can [illegible] to support a coherent global security [illegible]y, and can [illegible] uated as true distributed systems.

[illegible] out complex systems [illegible] evaluation classes. [illegible] be possible, [illegible] ate with a [illegible] connection, given [illegible] latter is within the [illegible] gles the mind to [illegible] rity policy that [illegible] he multilevel host. [illegible] h host granularity, [illegible] larity within each of the [illegible]

It is our [illegible] complex system should not be evaluated a[illegible] ted system, with an overall TCSEC evaluation. In[illegible] should look at it as a collection of hosts with [illegible] a separately evaluated network service. Under these circumstances, the appropriate goal is to examine the individual host and network evaluation ratings, in order to justify continued accreditation of the hosts for their current mode of operation, in the face of their attachment to the network.

The environments guidance document associated with the TCSEC, called the "Yellow Book" [4], addresses the relation between the evaluated rating of an ADP system and the range of classifed information it can handle, on the basis of characteristics of its environment, such as the minimum clearance of users. An analogous document addressing the issues associated with connecting a host to a network is currently being developed by the NCSC with support from MITRE.

THE CASCADING PROBLEM

An example of an accreditation issue that needs to be considered in a complex system context was brought up by Steve Walker. Suppose that two ADP systems are operating in controlled mode at two adjacent security levels, but one has the range TS-S and the other has the range S-C. They could be connected by a trivial network consisting of a single, physically protected wire joining S-level ports on both systems. The problem is that the network connection has created a risk of introducing TS information into the C-S system, whose accreditation only qualifies it to handle the two lower levels.

From the point of view of the TCSEC, the network connection has merely introduced a single-level-S resource to both hosts. No new software has been added to either host, so their evaluation classes have not been affected.

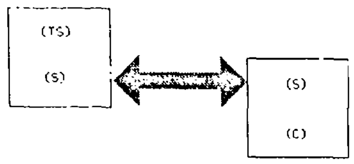


Figure 7 The Cascading Problem

What went wrong? Evidently, the environment of the hosts changed by connecting them to the network. We could say that the set of human users was expanded, but there is a more precise way of characterizing the problem, relating to the trustworthiness of security labels placed by a computer system on classified objects. In general, the object level is determined from two influences:

Object Level = Source Level + Contributions

When information enters the system from outside, the security level of its source is known and trusted. Thereafter, while information is held within the system, the correct level is maintained by system software. When computations cause information to flow into an object from another, access controls ensure that the level of the object remains consistent with the level of information contributed to it by those computations.

The TCSEC rating of a system is a measure of the trustworthiness of system software in maintaining object levels during computations; but how trustworthy is the determination of source level? In a standalone ADP system environment, the normal external source of information is human users. If a human user says that certain input information is Secret, high confidence may be placed in that assignment. For, if the user is only cleared to Secret, he does not have any higher-level information to introduce; and if he is cleared to a higher level, he can be trusted to give the proper level for information at that level or lower.

When an ADP system is connected to a network, the network becomes a new source of information, and it often cannot be trusted to the extent that human users are. This is one point that must be taken into consideration when writing an Environments document for networks, or a distributed system evaluation guide.

A similar problem can occur even for a standalone system. Again, consider the two controlled-mode systems, one at TS-S and the other at S-C, but do not connect them. Can we make a tape on the higher-level system with S-level files on it, and carry it to the lower-level system? No, because the tape is an external source of information, and its security label, determined by the other controlled mode system, cannot be trusted any more than if the information came across a wire. Such a tape transfer would be permissible only if a responsible individual has reviewed the material on the tape and confirmed the correctness of its marking, or if the tape was produced on another system that did not handle higher-level information.

Problems like this can be solved by imposing additional restrictions on interconnection. For example, as Walker has suggested, one can insist that all mutually connected systems operating over the same size security level range (two adjacent levels, in the example) share the same maximum level.

When an accrediting agency wishes to place more severe restrictions on certain information than called for by normal environmental guidelines, the natural approach would be to set up a community of hosts satisfying the tighter restrictions. Communities like this can be implemented by discretionary access controls or encryption.

4. SUMMARY

- Protocol layering is important in network architecture, and it has consequences for network security evaluation. A network is viewed as a global service provided by the user interface to its outermost protocol layer.

- In attempting to use the TCSEC to evaluate a network, a rough strategy is to apply security policy requirements to the network globally, and architectural requirements to the network components.

- Network global security policies should be stated in terms of concepts supported by a particular protocol layer. Requirements on more than one layer may be called for. The global policy has implications for derived functional requirements on individual components, to support it.

- Examples of networks providing varying features and levels of assurance have suggested that the use of encryption should be regarded as an architectural feature of a network, affecting the evaluation class.

- Separate requirements documents or appendices should be published for specific types of network components. In particular, it should be possible to consider entire subnets as network components. TCSEC requirements need radical reinterpretation for application to components, so that they do not exclude, or place unreasonable requirements on, specialized components or subnets. Component evaluation could assign separate assurance levels to various features appropriate for the component.

- A true distributed system has a global user interface whose security policy can be evaluated by the TCSEC.

- Complex distributed systems consisting of dissimilar hosts are not practical to evaluate as true distributed systems. Instead, the goal of evaluation for such systems is twofold: to evaluate the network itself, and to justify continued accreditation of the hosts for their current mode of operation after attachment of the network. An environments document is needed to facilitate this. The fact that a network brings new, less trusted sources of information to a host is an important environmental consideration.

REFERENCES

1. "Proceedings of the Department of Defense Computer Security Center Invitational Workshop on Network Security," New Orleans, LA, March 19-22, 1985.

2. DoD Computer Security Center, "Department of Defense Trusted Computer System Evaluation Criteria," CSC-STD-001-83 (the "Orange Book").

3. Tanenbaum, A. S., *Computer Networks* , Prentice-Hall, Inc., Englewood Cliffs, N. J., 1981.

4. DoD Computer Security Center, "Computer Security Requirements," CSC-STD-003-85 (the "Yellow Book").

This work was supported by the U.S. Government under contract no. F19628-86-C-0001.

"Smart" Terminals for Trusted Computer Systems

by Mark D. Gabriele

Abstract

"Smart" terminals are increasingly popular, as they can increase individual productivity immensely. However, such terminals are not presently desirable from the point of view of building a secure multi-level computer system, as they open avenues for spoofing, covert channels, and relabeling of sensitive data. This paper is an overview of the problems and the possible solutions to the problems created by using "smart" terminals in trusted systems. Among those solutions are: 1) don't use smart terminals; that is, restrict trusted systems to "dumb" terminals exclusively; 2) use only terminals which are "configurably dumb;" 3) alter existing "smart" terminals to remain "smart" while becoming "trustable;" and 4) use secure workstations as "smart" terminal emulators. Each is examined and weighed.

Introduction

The user community has recognized a need for some method of accessing secure systems which will increase individual productivity. This is accomplished on non-secure systems by the use of "smart" terminals. This paper will focus on what types of terminals may be used for accessing secure host systems without jeopardizing their security. Perhaps some of the types of secure terminal mentioned here will be researched and developed, and eventually integrated into the secure systems of the future

These several generic types of terminal range from "dumb" to "smart" to the secure workstation of the future. The advantages, drawbacks, and security relevant aspects of each will be discussed. Methods for securing each type of terminal will be suggested, as well as possible problems which may need to be overcome. The end result will be that the reader will have some idea about the state of secure terminals today, and where they may be going in the future.

There are some matters which are not addressed in this report. The foremost is emanation security (the Tempest problem). Other exceptions will be mentioned as they occur.

Disclaimer: The views expressed in this paper are exclusively those of the author based on experience gai-ed as a commercial products security evaluator at the National Computer Security Center (NCSC). This paper does not necessarily represent official policy of the National Computer Security Center.

Terminology

Before beginning this discussion, a number of definitions are in order. First, we need to define our conception of a "smart" terminal:

> A "smart" or "intelligent" terminal is a terminal which possesses some form of volatile or non-volatile programmable memory, and allows the host system to perform read and write operations on the data in that memory.[1]

In contrast, a "dumb" terminal has no programmable memory. A "configurably dumb" terminal is a unit which may have unlimited data processing and storage capabilities, but these can be disabled to render the machine "dumb," according to the above definition.

A "trustable" terminal is considered to be a device which can be relied upon to relay to the user exactly what was received, transmit exactly what the user entered for transmission, and to provide separation across all security levels.[2]

With these definitions as a basis, there must now be a distinction made between what constitutes a terminal and what constitutes a network node. If such a distinction is not made, then one can simply argue that any intelligent terminal attached to a host constitutes a network, and should be dealt with as such from a security standpoint.

An explicit definition of "network node" is needed to alleviate this problem. Owing to the increasing complexity of computer networks, a node is a difficult thing to characterize; but for the purposes of this paper, a node is:

> "A device which provides CPU cycles in support of some activity which is invoked at some other point on the network."

Where a network is simply defined as an interconnection of two or more nodes.

Note that while a personal computer may physically be able to comply with the above definition, should this capability be neutralized or defeated by some mechanism, then that unit is no longer acting as a node. As an example: if a personal computer is running a communications package which includes a file transfer protocol, that machine is acting as a terminal, not a node, until such time as the host requests that file transfer protocol is activated and the machine enters server mode. At that point, the personal computer is providing support to a remotely activated activity (file transfer, in this case), and is considered to be a node.

---

[1] As appeared in response 147 in the DOCKMASTER computer system Criteria Discussion forum, entered by Vidmar.CPE.

[2] Ibid.

Dumb Terminals

The first configuration which will be addressed is that of "dumb" terminals. These are secure simply because they have no means by which they could compromise or subvert sensitive data. This type of terminal is exemplified by the generic term "glass TTY," although many types of printing terminals would certainly qualify. A truly dumb terminal would include no buffer memory, although many contemporary terminals which are considered to be dumb do contain some memory. Just because a terminal is considered to be dumb does NOT mean that it must be inconvenient or cumbersome to work with; however, any "intelligence" which the terminal exhibits must be provided by the host machine. This requirement severely limits the utility of a dumb terminal. All dumb terminals suffer from similar problems, to varying degrees, regardless of their apparent intelligence at their user interface.

One drawback is that of independence of components. When working with a dumb terminal which must rely on a host for even the simplest of chores, all work must be done while the user is on-line with the host. This creates dependence on one central host; should that host fail, or suffer from poor response time, the user is unable to work. Entire offices or even corporations can be stymied by a host failure; if all computing is done via dumb terminals, NO work can be done on the terminal until the host service is restored.

Hosts which support a generous user interface on a dumb terminal may unfortunately be slowed by processing delays. The host processor may incur a great deal of overhead doing menial, terminal-support tasks, slowing system response; again, user productivity suffers. Even the best dumb-terminal systems have these faults.

Examples of very popular dumb terminal systems may be found in many configurations of the IBM 327x series of terminals and terminal control units. The 3278 terminal supports very limited local functionality: basically, only the ability to position the cursor, and send up to one screen of information back to the host at a time. Virtually no processing of the data is done locally; although there is some slight local intelligence, the 3278 terminal is essentially dumb. The 3274 (and related type) device controller, while supporting error detection and correction, does not add to the local functionality of the terminal. Almost all terminal support, such as buffered screen memory, various screen set-up options, etc. must be done by the host. The host software support for the terminal must therefore be trusted code in order for this configuration to be considered secure. Even though this arrangement does provide some of the functionality of a smart terminal with few security-relevant drawbacks, it is obvious that it is not the most economical method in terms of host CPU time, in addition to the disadvantages listed above.

All NCSC-evaluated systems require the use of "trustable" terminals in their evaluated configurations. Dumb terminals are considered intrinsically safe because they cannot aid a malicious user in attacking the system by any known means. They are therefore defined to be "trustable". They also tend to offer fewer features than contemporary computer users would like. However, at the present time there are no "trustable" smart terminals. So, the user of a secure computer system must currently use a dumb terminal in order for the system to remain secure.

Configurably Dumb Terminals

The modern user of a secure system, in order to have his system running in its evaluated configuration, may need to have two separate devices on his desk: a terminal for communications with the host machine, and a personal computer for use with spreadsheets, word processors, etc. This tends to be an impractical, as well as an inconvenient solution, so in many instances, a personal computer (PC) may be used as a terminal device. This is normally accomplished by running some type of terminal emulation software. Regardless of the software being run, this is almost never a secure configuration. Too many possibilities of subversion exist: the PC can "spoof" a user into divulging his or her password, keep a record of the entire conversation with the host for later retrieval by another party, open enormous covert channels, relabel sensitive data, or destroy any trusted path which may exist. Unfortunately, great numbers of PCs are being used as terminal emulators; so some action should be taken to render them secure enough to be used as trusted terminals.

The path by which this may be done is to render them "configurably dumb." What this means is that the user may invoke some action which causes the PC to lose those things which make it untrustable. As an elementary example, one may install an extra processing card in a generic PC which, when activated, causes the machine to reboot from a trusted ROM on that card, and immediately execute a trusted terminal program, also contained in ROM. When the card is active, the personal computer functionality of the PC is gone; it is only capable of acting as a terminal. That terminal will be trusted at the level of the software and hardware modifications. It should be a goal of the NCSC to develop component evaluation criteria which can address machines of this ilk, as they would allow the user to fashion his PC into a trusted terminal. This terminal could be either smart or dumb. If it is to be made smart, then it will be covered by the discussion of smart terminals which follows; if it is to be made dumb, then it must exhibit none of the functionality of a PC; it must be trustable in exactly the same manner as any other dumb terminal. Note that switchably rendering an expensive computer incapacitated except for basic terminal emulation functions may sound somewhat ludicrous; but if a dumb terminal is all that is needed, it may be more economical to arrange to equip a PC with such a device, so that it may serve both stand-alone and terminal emulation purposes equally well.

Smart Terminals

An alternative to the use of a dumb terminal in a secure computer system is to employ a trusted smart terminal device. This is a very favorable alternative in many cases because of the great functional enhancements which many smart terminals incorporate. Some are able to do high resolution graphics, while others allow great ease in manipulation of text, several pages of conversation buffer, multiple concurrent terminal sessions, or even multiple sessions on different machines which are physically plugged into the same terminal. A few terminals allow all of these things and more. Needless to say, these devices can increase the productivity of the mainframe user immensely, while presenting the user with a much nicer machine interface. Apparently, everyone wins. Unfortunately, this is not true from the viewpoint of someone trying to secure a system which uses smart terminals for communication with a mainframe.

There are several features of smart terminals which can pose major threats to security. Foremost among these are: the threat of spoofing, the ability to relabel sensitive data, the ability to open extremely high-bandwidth covert channels, and the ability of such a terminal to interfere with a trusted path. Object reuse can present a readily exploitable threat. Each one of these flaws could be used to compromise sensitive data across all levels of the trusted computing base (TCB).

The spoofing attack could be employed by writing a program which runs on the smart terminal device. This program simulates a successful connection to the host machine and a logon banner. The program then prompts the user for their password, and stores the password for later retrieval by some malicious user. The attack is identical to the classical spoofing attack, but is carried out by the terminal rather than the host. This can make it more difficult to locate the perpetrator. This problem goes hand in hand with the problem of trusted path, which is not addressed by the Department of Defense Trusted Computer System Evaluation Criteria (TCSEC) until the B2 level. Once one has a trusted path, a spoofing attack from the terminal level is no longer a problem; however, in a smart terminal which features user-programmable keys, the "secure attention" key may be reprogrammed by a malicious user to destroy trusted path and allow a spoofing attack to take place. Thus, the secure smart terminal must have at least one key - the "secure attention" key - which CANNOT be reprogrammed. This key should send some unchangeable signal to the host, which the host must interpret as a request for trusted path establishment. In addition, the terminal must have no way of generating that signal except via the "secure attention" key.

A smart terminal may also have some "conversation buffer;" that is, some memory of the transactions between the user and the host. In many systems, everything the user inputs and everything the host machine outputs is saved, up to the limits of memory included in the terminal. In this conversation buffer there is great potential for subversion of data. The user password may be saved in plaintext, or any sensitive information which the user may have been working with may be able to be recalled by the touch of a single button. This can be a great convenience and enormous time-saver to the legitimate user, but if that user logs off and leaves his or her terminal without clearing the terminal's memory then the problem of object reuse occurs. The object is, in this case, the terminal's memory; this memory must be cleared between users, so that there is no possibility that one user can get at another user's data. The clearing of memory must therefore take place after the termination of each terminal session, as well as any other time where failure to do so could violate system security policy, such as downward level changes.

Since any smart terminal must have some ability to locally process data, another attack may be effected. This one is substantially more difficult and intricate than those mentioned thus far, but is certainly as compromising. If the terminal software in a smart terminal is modified by a malicious user, the terminal could be used to relabel sensitive data by intercepting and modifying input lines according to its programming, allowing it to downgrade or otherwise compromise sensitive data. This is a classic example of a "Trojan Horse" attack, applied through the use of a terminal.

The final method of attack which will be detailed here applies only to a terminal which supports multiple concurrent terminal sessions, either on one host or across many hosts. This is the problem of covert channels. Covert channels have long been recognized as a means of downgrading sensitive data on a host system, and could be used to downgrade information on a terminal as well. On a mainframe, the covert channel is often related to monitoring of use of system resources. In a smart terminal, a covert channel could take the form, for example, of the use of ACK and NACK signals between the terminal and the host, each signaling to another concurrent process either a one or zero bit of information. This is a simple operation, but an effective one nonetheless.

Regardless of all of the possible attacks which might be made on a computer system through the use of a smart terminal, the risks are not insurmountable. All of the above security weaknesses which smart terminals may exploit can be done away with in properly designed and installed smart terminal devices.

The major problem revolves around trusted path. If the user can be assured that he or she is in contact with trusted software at both the host and the terminal, many of the opportunities for defeating the security of the terminal can be removed. All trusted path mechanisms require the physical integrity of all devices which are part of the trusted path. This is readily accomplished at the mainframe level, but can be difficult to assure at each terminal, particularly when terminals are distributed throughout a complex. One method is to seal shut the casing of the terminal with some protective and unforgeable seal; this seal would show any sign of tampering, and users would be instructed not to use any terminal which had been tampered with, and report it immediately. Physically locking down the terminal in a manner in which it could not be easily tampered with is another solution. One major objective of either of these methods is to insure the integrity of the secure attention key, which would generate a non-maskable interrupt to both the host and the smart terminal, and guarantee to the user that the software at both ends of his or her connection was trusted. The other major objective of physical protection of the terminal device is to insure the integrity of the terminal's trusted software. This software is often ROM-resident, and with the replacement of a single chip, a malicious user could compromise the entire system.

One example of a way to cut down on the amount of trust placed in a ROM-based terminal program in the smart terminal is to cause the terminal program to be downloaded from the host when the user hits the secure attention key. Assuming the integrity of the secure attention key, this provides the user with good assurance that he is using trusted software; it also allows upgrades to be made to the terminal program very easily, and much less expensively than replacing the ROMs in every smart terminal at the installation.

The problem of object reuse in a smart terminal can be partially solved by erasing the conversation buffer as soon as the connection to the host computer is terminated. This may be accomplished by instructing the hardware or the firmware in the smart terminal that the conversation buffer is to be emptied, say, every 10 seconds if the terminal is not connected to a host. The terminal may also be programmed to erase the conversation buffer upon receipt of a given signal from the host. This signal would then be sent at any time the conversation buffer should be cleared (e.g. downward level changes). These instructions should be encoded in hardware or firmware so that they cannot be defeated by the user reprogramming the smart terminal in the course of his or her terminal session.

Relabeling of sensitive data may be seen as an extension of the trusted path problem. If the user is assured that he or she is using trusted software, then relabeling is no longer a problem, because the trusted software will not allow it. Covert channels also become no threat, provided that the trusted software takes measures to insure that they are rendered harmless. What is crucial is that the smart terminal software be trusted, and that the user be able to confirm that he or she is actually using the trusted software at any given time.

Since the major threats caused by the use of a smart terminal have been addressed, the question becomes one of proving that a given terminal device is trustable. In the case of a smart terminal, different threats can be mapped to different levels of trust in the <u>Department of Defense Trusted Computer System Evaluation Criteria</u>. The TCSEC does not address terminals as such; but by mapping the applicable Criteria requirements to terminal devices, it may be possible at some point in the future to define "levels of trust" within the realm of terminals and terminal emulation programs. One could then speak of a "B2-trustable" terminal, for example. Such a terminal would meet B2-level requirements for all relevant features, among which would be object reuse, covert channels, mandatory flow policies, and trusted path. A B2-trustable terminal would also require such things as B2-level configuration management and design documentation. It would, however, omit requirements which do not apply to a terminal device, such as discretionary access controls and auditing. A terminal of this sort could be used to run concurrent sessions at multiple levels (say, Secret and Top Secret) and be trusted to enforce the mandatory flow policies of the system, depending upon the level of trust bestowed upon the terminal.

If this methodology were to be uniformly applied, it could be said that a C2-level smart terminal - one which handled object reuse and some spoofing problems - could be connected to any C2 system which could be adapted to handle its special protocols, etc. without placing the system in grave danger of compromise. The same could be said of systems at any level of the Criteria; if we have a B2-level terminal device, then it should be trusted enough that we can connect it to not just one but two or more B2-level hosts which fall within the same range of trust, and rely on our terminal device to maintain the integrity of data labels. This involves placing a great deal of trust in terminal devices. To this point, the NCSC has not evaluated them; however, this will have to change if the NCSC wishes to provide its clientele with an Evaluated Products List (EPL) full of modern and user-friendly equipment.

Secure Workstations as Smart Terminals

Perhaps the optimal solution to the need for secure smart terminals may be solved by the use of the forthcoming secure workstation as a smart terminal. This gives the user the best of both worlds; the power of a mainframe when needed, with the convenience of smart terminal features, and the ability to do stand-alone processing for those jobs where a secure microcomputer workstation will suffice. A configuration such as this also makes possible many useful and security-relevant events which require some analysis.

To begin with, in order to rely on and trust the terminal software of the secure workstation, we must include it in the Trusted Computing Base (TCB) of the workstation. This will allow the terminal software to be trusted at the same level as the workstation. That is, a B2-level workstation may possess up to B2-level smart terminal trustability; it could therefore be used as one would use a B2-level smart terminal. In addition, when not in use as a terminal, it would retain the functionality of a secure workstation, within certain limits.

One important limit would be caused by the range of trust which is given to trusted computer systems In the example given above of a B2-level trusted workstation, the terminal software could be trusted up to a B2 level. Thus, the secure smart terminal/workstation as a whole would have a range of trust identical to a B2 system. If the machine were connected to a host (or hosts) which contained Confidential and Secret information, and the workstation was used to process Unclassified and Confidential information locally, the range of information accessible by that machine would span the range of Unclassified-Secret. That range is too great to be entrusted to a B2-level trusted system, according to the Computer Security Requirements document. Care must be taken that systems of this type are not inadvertently trusted beyond what can reasonably be expected from them.

It is also important to realize that a host cannot be considered secure at a level higher than that of its lowest terminal or workstation, unless the entire configuration has been specifically evaluated and it has been shown that that is the case. A B3-level trusted host may be subverted through use of covert timing channels on a B2-level trusted workstation, for example. Basically, all of the security problems which may plague a smart terminal exist for a secure workstation running smart terminal emulation software. Any further problems relate to the addition of some form of permanent storage in the secure workstation. If the smart terminal emulator takes advantage of the abundance of storage (typically several megabytes of hard disk) to provide additional features for the uploading and downloading of data, extreme care must be taken that the security policies of the system cannot be violated through its use. Again, the trusted terminal software will have to be evaluated by the NCSC along with the rest of the workstation in order to provide assurance that the security of the system will not be compromised.

Conclusion

It is obvious that a smart terminal can greatly increase the productivity of the typical mainframe user. It is also obvious that a smart terminal can significantly jeopardize the security of its host machine through many and varied mechanisms. However, these risks can and should be overcome if the user community is to be expected to switch over to using secure computer systems. If presented with an ergonomic and pleasant user interface, the user will not have to sacrifice efficiency and ease of use in order to use a secure system rather than a non-secure system. This should increase user acceptance of secure computer systems dramatically. Since it is imperative that both government and industry implement the use of secure computer systems, it is only logical that a comfortable user interface be provided. The use of smart terminals in secure computer systems can provide this interface, and perhaps encourage many hesitant prospective users to "go secure."

Bibiliography

Brotzman, Robert L. Computer Security Requirements -- Guidance For Applying The Department Of Defense Trusted Computer System Evaluation Criteria In Specific Environments. CSC-STD-003-85; Library No. S-226,727. 25 June 1985.

Latham, Donald C. Department Of Defense Standard Department Of Defense Trusted Computer System Evaluation Criteria. DOD 5200.28-STD; Library No. S225,711. December 1985.

Database Systems and the Criteria: Do They Relate ?

Erian S. Hubbard
Lt. Swen A. Walker
Ronda R. Henning

National Computer Security Center
9800 Savage Road
Fort Meade, MD 20755-6000
(301)859-4488

ABSTPACT

There is much debate in the computer security community as to whether or not the Department of Defense Trusted Computer Systems Evaluation Criteria (the Criteria) can be appli[illegible] to database management systems. In this paper we will examine the basic control objectives of the Criteria and the fundamental security concerns of database management systems. We will compare the two and show that, while the control objectives of the Criteria are applicable to database management systems, they do not encompass all of the security concerns in database management.

The views and opinions expressed in this paper are those of the authors and do not necessarily reflect official National Computer Security Center positions.

INTRODUCTION

The need for secure database management systems stems from the fact that, within the Department of Defense (DoD) and in corporations around the world, there is an increasing amount of information being manipulated through database management systems. The databases usually contain some classified or otherwise sensitive information, forcing these systems to operate in a system-high or dedicated mode. A user may need to access data of differing classification levels at the same time: consequently, data must be duplicated on separate machines for different security levels and compartments. This duplication of data on separate machines causes inconsistencies in the data. There is an urgent need within the DoD to replace these systems with multilevel secure systems. Additionally, other commercial customers such as financial institutions, would also be able to take advantage of the protection these systems can offer.

Computer security research and development began in the late 1960's. The earliest work concentrated on the area of multilevel secure operating systems with database management security research and development receiving relatively little attention. One of the main reasons for this lack of attention was the perception that one could not credibly implement a secure database management system which was dependent on the security controls of an untrusted operating system. At that time, secure operating systems were, for the most part, nonexistent. Since then, a few multilevel secure operating systems have been developed by commercial vendors; however, a secure multilevel database management system still does not exist. Present day database management systems do not provide adequate security controls and mechanisms to ensure that users are allowed to access only that data for which they have been granted a clearance and have a specific "need to know."

A major conclusion of a 1982 Summer Study on "Multilevel Data Management Security"[1] was that computer security technology had advanced to the point where certifiable multilevel database management systems could be built for several specific applications in three to five years. However, there is no metric to evaluate "secure" database management systems against. It has been proposed that the DoD Trusted Computer Systems Evaluation Criteria[2] (the Criteria) is sufficient as a database management system security criteria. We do not subscribe to this view.

In this paper we will examine the basic objectives and requirements of the Criteria to discover where they may fall short of the requirements for security in a database management system. The views expressed in this paper are those of the authors and are not intended to be taken as policy. This paper is an attempt to raise the readers awareness of the issues vital to database security that have not been adequately addressed.

CRITERIA

We begin by pointing out that, when the Criteria was published in 1983, it was defined to apply to both trusted general-purpose and trusted embedded systems, not for direct application to database management systems. With that fact in mind, the Criteria was developed for a number of reasons:

° To provide users with <u>a metric with which to evaluate</u> the degree of trust that can be placed in computer systems for the secure processing of classified and other sensitive information.

° To provide <u>guidance to manufacturers</u> as to what security features to build into their new and planned, commercial products in order to provide widely available systems that satisfy trust requirements for sensitive applications.

° To provide <u>a basis for specifying security requirements</u> in acquisition specifications.

In order to meet these goals, the Criteria sets forth three basic control objectives which are concerned with security policy, accountability, and assurance.

The first of these, the security policy control objective, requires that a statement of intent with regard to control over access to, and dissemination of information must be precisely defined and implemented for each system that is used to process sensitive information. The security policy must accurately reflect the laws, regulations, and general policies from which it is derived.

In systems processing classified or other specifically categorized sensitive information, provisions must be included for the enforcement of mandatory access control rules. These provisions must include a set of rules for controlling access based directly on the comparison of an individual's clearance or authorization for the information and the classification or sensitivity designation of the information being sought. These rules should also control access based indirectly on considerations of physical and other environmental factors of control.

Systems designed to enforce a mandatory security policy must store and preserve the integrity of classification or other sensitivity labels for all information. Labels exported from the system must be accurate representations of the corresponding internal sensitivity labels.

These systems must also include provisions for the enforcement of discretionary access control rules. That is, they must include a consistent set of rules for controlling and limiting access based on identified individuals who have been determined to have a need-to-know for the information.

The accountability control objective requires that systems processing or handling classified or other sensitive information must assure individual accountability whenever either mandatory or discretionary security policies are invoked. Futhermore, to assure accountability the capability must exist for an authorized and competent agent to access and evaluate accountability information by a secure means, within a reasonable amount of time and without undue difficulty.

The assurance control objective requires that systems processing or handling classified or other sensitive information must be designed to guarantee correct and accurate interpretation of the security policy and must not distort the intent of that policy. Assurance must be provided that correct implementation and operation of the policy exists throughout the system's life-cycle.

We believe that, for the most part, these control objectives have a great deal of merit when put in the context of database systems. However, they are not quite enough to cover all of the concerns that are faced when attempting to develop a secure database management system. We must consider data integrity, inference, aggregation, and many other problems not addressed in the Criteria. We must also expand on the concepts of labeling and auditing when dealing with database systems.

EXAMPLE

In order to make the security concerns associated with securing a database management system more evident, we will use the sample database shown in Figure 1 to provide examples of the issues discussed below. The sample database will cor[illegible] of personnel information. The databa[illegible]ord will contain the employee's name, [illegible] security number (ssn), sex, salary, a[illegible] department.

NAMF	SSN	SEX	SALARY	DEPT
John	123456789	M	50000	A
Ronda	268034721	F	25000	B
Brian	106638528	M	17000	C
Larry	186539679	M	35000	A
Bruce	873595357	M	44900	B

FIGURE 1.

DATA INTEGRITY

In the Criteria's control objectives, integrity is only discussed as it relates to sensitivity labels and system integrity. For database management systems, we must extend the notion of integrity to address the issues of accidental or unauthorized modification of data and integrity checking for the accuracy or correctness of data within the database. The first integrity issue is that some system data may need to be viewable by all security levels but only modifiable by certain trusted programs or authorized users[3]. This is exemplified by the case of a user examining the sample database for the first time and wanting to view the structure of the record in the system catalog. We want him to have access for examining the structure of this table but not access for modifying it. We would want this access to be regulated through a mandatory policy. The mandatory policy of the Criteria only addresses the improper disclosure of information, not its modification. An integrity policy requirement is needed to enforce the prevention of unauthorized or unintentional modification or destruction of data or other essential, database-related information. It must be precisely defined and implemented for each system processing sensitive information and must work in concert with the system security mechanisms and controls.

In order to ensure that the data in our database remains correct "integrity constraints" must be imposed on individual transactions being performed on our database. A simple example of an integrity constraint for the sample database in figure 1 would be that all sala[illegible] values must be greater than zero. The data integrity problem is exemplified in the above database when a user wishes to change the department of John to Z. Assuming that the user has authorized discretionary access rights, the issue to be addressed is whether or not the value to be placed in the field is an acceptable value. In other words, does a Z department exist within this organization? Another problem that can arise from the lack of data integrity controls is that a user may be [illegible] to write a large quantity of false or incorrect data to the above database, rendering any queries on this database useless. Although some commercial systems do provide some integrity checking, most integrity constraints are weak or nonexistent[4]. Most integrity checking today is still done by user-written procedural code executed outside the control of the database management system.

As mentioned by Date, many systems claiming to provide data integrity are actually using the term to mean concurrency control. Systems that provide "integrity" in this sense typically guarantee only that interference between two concurrently executing transactions cannot occur; they do not concern themselves with the question of whether individual transactions are correct in themselves.

Under the Criteria's extension of the assurance control objective, there is a requirement for "System Integrity." The system integrity requirement states that: "hardware and/or software features shall be provided that can be used to periodically validate the correct operation of the on-site hardware and firmware elements of the Trusted Computing Base (TCB)." Does this requirement have any application in the database management system world, or is it sufficient to rely on the operating system to handle system integrity? Does this requirement apply to software releases of database management systems, or only hardware modification?

INFERENCE

The inference problem is defined as the compromise or increased probability of compromise by deduction of unauthorized information due to combinations of the possession, known existence, known absence, chronology, and location of authorized information. It may be considered a covert channel of database management systems. It is a serious problem that must be reconciled before a database management system can be regarded as secure. As an example of the inference problem, consider the following: if the quantity (q) of some manufactured item is classified, then either the total production budget (b) or the cost per item (c) must be classified, since quantity (q) could be derived as q = b/c.

There are many unanswered questions in the area of inference control. For instance, can the problem be addressed or quantified in a general way, or must it be addressed case by case for each site processing sensitive data or each specific application? Should metrics be developed to describe and quantify an acceptable level of inference control? Should we require that abstract tools be provided in a database management system so that inference control can be builtin at each site? If inference is actually another covert channel, should it or could it be audited in the conventional sense of the Criteria?

Since it is unlikely that the general algorithms can be defined for limiting the inference problem, we believe it would be wrong to require that a complete solution to the problem be built into a database management system. However, metrics should be developed to define acceptable bandwidths of possible inference attacks, and mechanisms should be required to be available within a system which will allow the inference problem to be restricted to an acceptable level. These restrictions could then be implemented by each site as the need arose. The audit mechanism must also be able to audit possible inference attacks.

AGGREGATION and CONTEXT

The classifications assigned to data must account for the data's associations or relationships with other data. For example, the unclassified data describing a flight may be classified when the flight itself becomes explicitly associated with a particular mission, cargo, or passenger.

Because classification can increase with context, an assemblage of data items may have a sensitivity far higher than that of an individual item in the assemblage. For example, the location of one particular submarine is likely to be less sensitive than the location of all submarines. Another example is that a single phone number may be less sensitive than a complete telephone directory.

Since classification depends on context, it is not enough to store labels with the physical data records in the database. Methods are also needed for determining the classification of data when it is associated with other data and for managing modifications in these associations. General algorithms for context classification and data aggregation may only be possible on a per application basis, it may not be possible to maintain these relationships on a general database management system level. However, mechanisms can and should be provided so that labels can be enforced on aggregated data once it is identified in a particular application.

LABELING

Labeling is an area in which the Criteria may fall short of as well as exceed the needs of database systems. It falls short in that there are no requirements for labeling according to the context of the data. It may exceed our needs in that a database system does not operate devices and, therefore, should be able to rely on the operating system for exportation of data to the proper devices. However, if the database system operates on data objects which are smaller than those which the operating system works on (e.g., the operating system may operate on a file, while the database management system operates on records within a file), an interface must be defined such that the lower levels of labeling can be supported and employed.

There are some very difficult issues which must be addressed in the area of labeling. If the labels are to be maintained at the entity level, is the data then labeled at the user's current security level, or can labels exist at the discretion of the user? How do we keep data from migrating to the user's highest classification and, thereby, having data which is over-classified? How are data labels maintained during rollback and recovery? How are labels affected by changes made to the data?

We believe that mechanisms should be required which allow data to be labeled to whatever granularity is required by an application and that the mechanism ensure the integrity of these labels. We also feel that mechanisms should be required which allow proper labeling of aggregated data.

AUDITING

Under the accountability control objective of the Criteria there is a requirement that "the TCB be able to create, maintain, and protect from modification or unauthorized access or destruction an audit trail of accesses to the objects it protects." Auditing is, of course, very important in database management systems; the issue is what do we audit? The types of events that should be audited are logon/logoff, creation/deletion/modification of objects, access to objects, and actions taken by database administrators and system security officers. This list is open to any additions, but we feel that this is the minimum set of events that should be audited in a database management system. For each of these events the audit record should identify the date and time of the event; user; type of event; success or failure of the event; the user's security level; level of the object accessed; and, where applicable, before and after image of the object. Since the objects of the database management system can be at such a fine granularity, the audit trail could become quite large very quickly and, therefore, quite useless without very sophisticated audit-reduction tools. It would seem reasonable to make auditing possible to the finest granularity level of the data but also allow the individual sites the discretionary control to audit whatever level of granularity would be most useful to them. The Criteria also requires that the "system administrator shall be able to selectively audit the actions of any one or more users based on individual identity and/or object security level." This is a very useful mechanism to have in place. However, in a database management system it would also be very useful to be able to selectively audit the actions taken on any object based on the operation and/or the user's or object's security level.

CONCLUSION

Because there is a great need for security in database management systems and the security requirements for various sites differ, it is very important to have a metric with which to evaluate the degree of trust that can be placed in database management systems. It is also very important to provide a basis for specifying those security requirements in acquisition specifications. However, in performing these evaluations, or when generating system requirements, we must consider all of the security relevant issues. Because the Criteria was originally defined to apply to trusted general-purpose and trusted embedded systems, the control objectives of the Criteria (while valid when applied to database management systems), are not quite sufficient to encompass all of the security concerns in database management system. We must consider the problems which have been discussed in this paper as well as any other yet-to-be-discovered problems in the area of secure database management systems. Only after these issues are discovered, fully understood, and dealt with properly can database management systems be considered secure.

REFERENCES

1. Committee on Multilevel Data Management Security, "Multilevel Data Management Security," Technical report, Air Force Studies Board, National Research Council, 1982.

2. DoD Computer Security Center, Department of Defense Trusted Computer System Evaluation Criteria, Fort George G. Meade, Maryland, 15 August 1983, CSC-STD-001-83.

3. Biba, K., "Integrity Considerations for Secure Computer Systems," ESD/AFSC, Hanscom AFB, Mass., April 1977 (NTIS AD 039324), ESD-TR-76-372.

4. Date, C. J. An Introduction to Database Systems, 4th ed. Reading, MA: Addison-Wesley, 1986.

TOWARDS PRACTICAL MLS DATABASE MANAGEMENT SYSTEMS USING THE INTEGRITY LOCK TECHNOLOGY

Rae K. Burns
The MITRE Corporation
Burlington Road
Bedford, Massachusetts 01730

This paper explores some practical considerations for using the integrity lock technology to provide multilevel secure (MLS) database management systems. A prototype architecture is described which minimizes the source code modifications necessary to retrofit the integrity lock mechanism into an existing database management system (DBMS). The INGRES relational DBMS is used to demonstrate the architecture. In addition to securing user-defined relations, the integrity lock software secures the INGRES data dictionary relations, thereby supporting classification at the record, relation, view, and index levels.

Funding for this work was provided by the U. S. Air Force Electronic Systems Division.

INTRODUCTION

The integrity lock technology has been demonstrated as a feasible near-term solution to the need for multilevel secure database management systems [1]. The work described in this paper derives from a current Air Force project to field this technology for use in a production environment. The questions are no longer ones of feasibility, but rather questions of a more practical nature. This paper addresses two such general questions:

1. How can the integrity lock be retrofit into a commercial off-the-shelf (COTS) DBMS without impacting DBMS maintainability? Could the integrity lock technology be used even if machine-readable source code were not available, or is access to source code a prerequisite for the use of the technology?

2. Can the integrity lock technology be used to address any of the database inference and aggregation issues? Can database views be secured with the integrity lock technology? How might secure views be implemented and used?

These two questions translate into the implementation goals for the INGRES prototype:

1. Implement the integrity lock technology with minimal changes to DBMS source code.

2. Use the integrity lock filter to secure the data dictionary and thereby extend the scope of data protection within the database to include relations and database views.

INTEGRITY LOCK TECHNOLOGY

The integrity lock concept is described in detail in references [2] and [3]. Basically, each record (or other database object) is tagged with its classification. Then an unforgeable cryptographic checksum for the entire record is computed and stored in the database with the record. This effectively "seals" the data, and any unauthorized modifications to the data or its security tag can subsequently be detected. In addition, access to individual records (or other objects) can be restricted based on some specific security policy. To implement the integrity lock mechanism, a trusted filter monitors the operations of an existing untrusted database management system.

Security Architecture

The integrity lock architecture divides the DBMS software into two separate executable components: one which interacts with a user and one which accesses the database files. All communication between the two components is monitored by a trusted software component which is independent of the DBMS software. The implementation requires three separate execution domains: the trusted monitor (FILTER), the portion of the DBMS which interacts with the user (USER), and the portion which accesses the data files (FILE). Figure 1 illustrates the interactions among these environments.

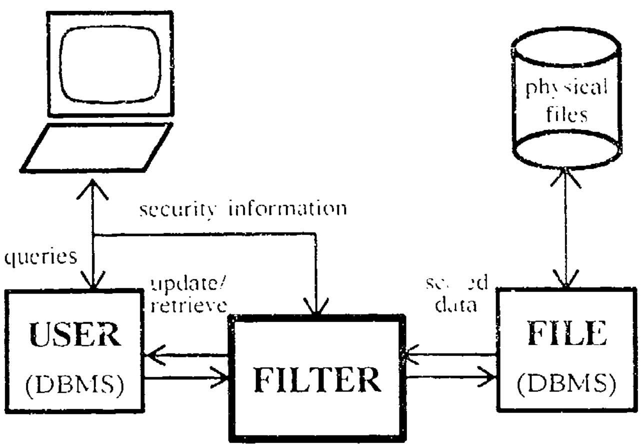


Figure 1. Integrity Lock Architecture

The operating system (or Trusted Computing Base (TCB)) within which these three components are executing must enforce some basic security requirements. The security characteristics of each domain are as follows:

FILE	The FILE component executes at the security classification of the database files. The FILE is the only executable module which has any access to the database files. It is prohibited from accessing any output devices or files at a lower security classification (i.e., a mandatory security policy is enforced by the TCB).
USER	The USER executes at the security clearance of the individual using the database. The USER is prohibited from accessing objects at a higher security classification and from accessing any output devices or files at a lower security classification, as specified by the TCB's mandatory security policy.
FILTER	The FILTER executes at the security classification of the database files. However, it is privileged to initiate the execution of the USER (at a potentially lower security classification) and to communicate with it. It also uses operating system trusted functions to determine the relevant security classifications and to audit security-related activities. The TCB has sufficient mechanisms to assure that the FILTER software is tamperproof.

The prototype was implemented within the context of the INGRES data base management system and the UNIX* operating system (BSD 4.2). The design makes use of several UNIX security-related features. However, since current implementations of UNIX are vulnerable in a number of areas [4], the implementation is not intended to be secure within existing UNIX environments.

Security Policy

The security policy for access to the information in the database is enforced by the trusted FILTER. For the purposes of the prototype, tuple (or record) level classification was used with a simple mandatory security policy based on the 1982 Air Force Summer Study [5]. The following is a summary of the policies (SC is security classification, SSO is a function which is true if the user is currently a System Security Officer):

```
READ Record     SC(user) dominates
                    SC(record)

APPEND Record   if SSO(user) then
                    prompt for SC(record)
                else SC(record) =
                    SC(user)

UPDATE Record   SC(user) dominates old
                    SC(record)
                if SSO(user) then
                    prompt for new
                    SC(record)
                else new SC(record) =
                    SC(user)

DELETE Record   SC(user) dominates
                    SC(record)
```

There are no other mandatory or discretionary access control policies for the prototype. INGRES supports some discretionary controls which may be used in addition to the integrity lock mandatory controls. Since the UNIX environment does not support security clearances for users, for this implementation, the applicable clearance is read from a ".secure" file within the user's home directory.

PROTOTYPE ARCHITECTURE

The primary goal of the INGRES integrity lock prototype was to implement the integrity lock mechanism without changing substantial amounts of source code. To achieve this, the INGRES object libraries were split along functional boundaries into two separate sets. The executable modules were then re-linked into the USER and FILE components of the integrity lock architecture. This section describes the

*UNIX is a trade/service mark of the Bell System.

FILTER interface which mediates access between the two sets of INGRES functions and provides an implementation methodology for developing an integrity lock version of a COTS DBMS.

INGRES Functional Interface

The INGRES system is highly modularized and contains a set of functions for low-level access to relations. These functions include relation open and close, get/put tuple functions, and supporting buffer management functions. The approach taken for the prototype was to use this relation access interface as the vehicle for the FILTER to mediate access to the database. Figure 2 represents the standard INGRES get_tuple function as it would be invoked by the INGRES query processor.

get_tuple (relation_desc, tuple_id, tuple)

where		
	relation_desc	the relation descriptor
	tuple_id	identifies a tuple
	tuple	a pointer for the returned tuple

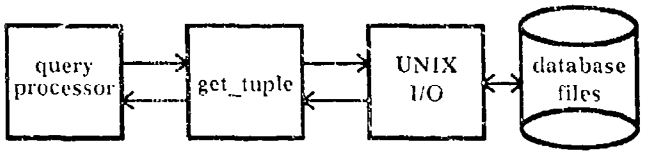


Figure 2. INGRES Function Invocation

For use with the integrity lock filter, the get_tuple function (along with all of the relation-level functions) is replaced by a substitute function which extracts the parameters from the call, inserts them into a message buffer, and communicates the buffer (via UNIX pipes) to the FILTER. The FILTER has an opportunity to perform any security processing before conveying the message on to the FILE process. The actual INGRES get_tuple function is invoked by the FILE process, and the tuple retrieved is passed back to the FILTER. Here the FILTER will recalculate the checksum and enforce the mandatory security policy prior to passing the tuple back to the USER process. Finally, the USER process will store the tuple into the location designated by the original get_tuple function invocation. Figure 3 illustrates this process at a conceptual level.

Development Methodology

For the operational version, only three INGRES source modules (out of a total of 346) were modified. In addition, only 2700 lines of additional C code (including 1000 lines of trusted code) were needed to implement the basic integrity lock

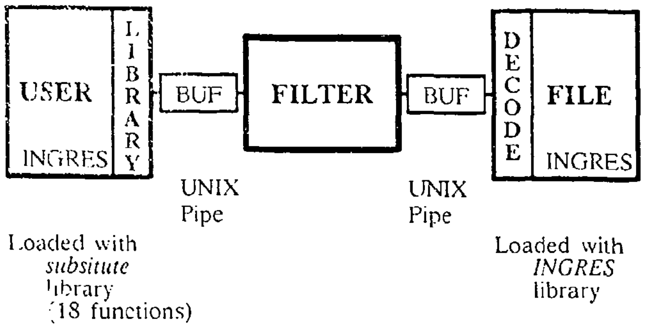

USER
INGRES
LIBRARY
BUF
FILTER
BUF
DECODE
FILE
INGRES
UNIX Pipe
UNIX Pipe
Loaded with *subsitute* library (18 functions)
Loaded with *INGRES* library

Figure 3. INGRES Process Architecture

functions. The construction process was done in five major phases. At the conclusion of each phase, there was a working version of the implementation up to that point. This technique allowed the problems encountered with each phase to be resolved prior to the introduction of more design detail. The five major phases were as follows:

1. Simple Prototype

   Simple versions of the USER, FILTER, and FILE processes were developed. These processes interacted to read and write lines of a standard UNIX data file. During this phase, the details of the inter-process communication were worked out and shown to be effective. Most of the issues which relate to the operating system environment were dealt with in this phase.

2. DBMS Restructuring

   This phase was spent researching the INGRES implementation, and dividing it into two pieces. Once the restructuring was complete, the INGRES software was integrated into the simple prototype from phase one. Using the resulting implementation, it was possible to verify that the correct arguments were being passed through the FILTER. The result of this phase was a working DBMS without any security features.

3. Security Processing

   The next phase was coding the security related functions and integrating them with the results of phase two.

4. UNIX Access Control

   The fourth phase was to develop an environment in which the UNIX access control features could be used to restrict the various processes to access only the appropriate files. If the UNIX operating system were robust enough to resist security penetrations, these access controls could provide a basis for secure MLS databases.

5. Secure Data Dictionary

The final phase applied the integrity lock technology to the INGRES data dictionary. The results of this stage are the topic of the next section.

SECURE DATA DICTIONARY

The final phase of the development integrated security processing for the relations which make up the INGRES data dictionary; these were not secured initially in order to simplify the implementation. There are six relations in the data dictionary:

relation	The relation relation contains a record for each relation defined in the database.
attribute	The attribute relation contains a record for each attribute of each relation.
tree	The tree relation contains parsed query trees which define database views.
protect	The protect relation specifies INGRES discretionary access controls.
index	The index relation contains a record for each index which has been created.
integrities	The integrities relation is used to specify any update integrity constraints.

By including these relations within the scope of the security processing, the descriptive elements of the database are tagged with a security level as they are created. In other words, if a relation is created when the database administrator (DBA), or other authorized user, is processing at the SECRET level, then the data dictionary records for the relation will also be tagged at the SECRET level. Similarly, views, indexes, and other system entities acquire mandatory security labels.

The primary result of this extension is that relations acquire a security classification independent of the level of the records within the relation. However, a user must have a clearance for the relation level in order to access any records within the relation. A second practical result is the ability to associate a security level with the definition of a database view.

Classified Views

A database view is simply a definition of a subset of the database, usually specified in the DBMS query language. When a user query refers to a view, rather than a relation, the result of the query is limited to those records within the view. (With the integrity lock, the result is further limited to those records for which the user has an appropriate clearance.) Views are frequently used to provide discretionary access controls, based on the content of the data records. For instance, a sales manager may be restricted to access only those sales records for his/her region. There are current research efforts to determine how views might best be used to provide mandatory access controls [6].

By providing for security labels for database view definitions, it is possible to limit users to views for which they have a clearance in addition to limiting them to individual records for which they have a clearance. Figure 4 illustrates how a join of two relations can be defined at a higher classification than the individual relations. (The range statement in the INGRES QUEL language associates a query variable, used in the where clause, with a relation or a view; it is similar to a from clause in SQL.)

```
In a SECRET session:

create Arelation (attrA1, attrA2 ... attrAn)
create Brelation (attrB1, attrB2 ... attrBn)

In a TOP SECRET session:

range of A is Arelation
range of B is Brelation
define view ABview (attribute sub-list)
     where A.attrA2 = B.attrB5
```

Figure 4. Classified View Definition

The ABview is a join of the information within the Arelation and the Brelation. The join operation is based on the values found in attrA2 and attrB5 that are equal in both relations. The use of the view ABview is limited to those users with a TOP SECRET clearance, independent of the level of the records within the view.

User Restrictions

The use of classified views requires two restrictions within the user environment:

1. A user may access only one view within any query. This eliminates the possibility of joining views. If users were allowed to join views together, additional inferences would be possible.

2. Only the database administrator has direct access to relations; all other users must access the information within the database only through pre-defined views. The database administrator defines those views by direct references to the underlying relations. The full power of the query language is therefore available only to the database administrator.

These two restrictions constrain the use of the query language to reduce the potential scope of inferences which can be made. They restrict users to only those specific views authorized by the database administrator.

Unresolved Issues

There are several uses of views which would be helpful for multilevel secure databases, but which are not supported by this concept of classified views. For instance, aggregations over an authorized view cannot be further restricted. It is not possible to classify the sum of the values of a particular field accessible by the view higher than the view itself. Similarly, this type of classified view cannot be used to sanitize information. The data within the view is tagged with its classification and will not be visible to any less cleared user even within the context of a pre-defined view.

The integrity lock technology is vulnerable to sophisticated Trojan horses within the untrusted DBMS [7]. This vulnerability remains an issue and the use of classified views introduces an additional Trojan horse threat. While the integrity lock assures that the classification of the view can not be altered, it does not automatically prevent the view definition (called a query tree) from being tampered with during the query processing. The query tree is a fundamental INGRES data structure and is, in fact, modified significantly during the query processing. The scope of the Trojan horse threat could be limited by placing portions of the query tree under the control of a trusted component.

CONCLUSIONS

Overall, the results of this effort have met the initial goals. In total, including the secure data dictionary, six INGRES modules were modified and recompiled. Two of these recompilations were due to mixed functionality within the original code. Both were initialization routines which affected several different functional areas; the modifications removed the code not related to the functions needed within the particular process. The third modification was to support index relations. Here, an assumption was made in the original code that the "tid" would be the last attribute in each tuple; however, with the addition of the security attribute, that was no longer true. One modification was made to support the creation of a secured data dictionary, and two modifications were needed to put the user restrictions in place. All other functionality was implemented within the integrity lock software itself.

The use of the integrity lock technology to secure the data dictionary extends the usefulness of the integrity lock approach while providing a necessary feature for any practical secure DBMS. The ability to classify views provides a foundation upon which to build a base of experience to determine how views can best be used to address mandatory access control needs in a database environment.

It is hoped that the techniques and processes developed for this implementation will be helpful in future work with the integrity lock mechanism and with other efforts to develop multilevel secure database management systems.

REFERENCES

1. Graubart, R.D., and Duffy, K.J., "Design Overview for Retrofitting Integrity-Lock Architecture onto a Commercial DBMS", Proceedings of the 1985 IEEE Symposium on Security and Privacy, pp. 147-159.

2. Graubart, R.D., "The Integrity Lock Approach to Secure Database Management", Proceedings of the 1984 IEEE Symposium on Security and Privacy, pp. 62-74.

3. Denning, D.E., "Cryptographic Checksums for Multilevel Database Security", Proceedings of the 1984 IEEE Symposium on Security and Privacy, pp. 52-61.

4. Ritchie, D.M., "On the Security of UNIX", Bell Laboratories.

5. Air Force Studies Board, Committee on Multilevel Data Management Security, MULTILEVEL DATA MANAGEMENT SECURITY, National Academy Press, 1983.

6. Denning, D.E., Akl, S.G., Morgenstern, M., Neumann, P.G., and Schell, R.R., "Views for Multilevel Database Security", Proceedings of the 1986 IEEE Symposium on Security and Privacy, pp. 156-172.

7. Denning, D.E., "Commutative Filters for Reducing Inference Threats in Multilevel Database Systems", Proceedings of the 1985 IEEE Symposium on Security and Privacy, pp. 134-146.

INTEGRITY IN TRUSTED DATABASE SYSTEMS

Roger R. Schell
Gemini Computers, Inc.
P.O. Box 222417
Carmel, CA 93922

Dorothy E. Denning
SRI International
333 Ravenswood Ave.
Menlo Park, CA 94025

INTRODUCTION

A trusted computer system is designed to be 'secure' with respect to some well-defined security policy. There are two major classes of information security policy: (1) secrecy policies which govern the disclosure of information and (2) integrity policies, which govern its modification. Although much of the literature on computer security emphasizes secrecy, for many systems integrity is of equal or greater importance. The DoD Trusted Computer System Evaluation Criteria[1] is careful to encompass (although not require) security policies that include integrity. A trusted computer system is designed to protect 'sensitive information,' which is defined in the Criteria as information that must be protected from "unauthorized disclosure, alteration, loss or destruction."

In databases, the term 'integrity' is interpreted broadly, as illustrated by the following definition taken from Date[2]:

> "The term *integrity* is used in database contexts with the meaning of *accuracy, correctness,* or *validity*. The problem of integrity is the problem of ensuring that the data in the database is accurate -- that is, the problem of guarding the database against invalid updates. Invalid updates may be caused by errors in data entry, by mistakes on the part of the operator or the application programmer, by system failures, even by deliberate falsification. The last of these, however is not so much a matter of integrity as it is of *security* ... The term 'integrity' is also very commonly used to refer just to the special situation ... in which it is possible that two concurrently executing transactions, each correct in itself, may interfere with each other in such a manner as to produce incorrect results."

In this paper, we address all aspects of integrity in that all are essential to the operation of secure database systems.

Classes of Integrity Policies

There are two distinguishable aspects of integrity policies: whether a given modification of information is *authorized*, and whether the modification results in information that is in some sense *consistent* or *correct*. Authorization is subdivided into two categories: (1) *mandatory integrity authorization*, which is based on integrity classifications, reflecting importance of data, and clearances, reflecting user trustworthiness, and (2) *discretionary integrity authorization*, which is based on users' needs to modify information. Both mandatory and discretionary integrity controls can protect data from malicious tampering and destruction as well as from accidental modification and destruction through operator errors (e.g., an operator may inadvertently attempt to delete the wrong relation) or faulty software.

Consistency is subdivided into three categories: (1) *database integrity rules*, which define correct states of a database in terms of relationships among the data, (2) *recovery management*, which returns the database to a consistent state after failure, and (3) *concurrency controls*, which ensure that concurrent transactions do not interfere, thereby creating inconsistent states of the database.

We shall discuss each aspect of integrity in more depth after first discussing assurance for these different aspects.

Assurance

The notion of a security perimeter is essential to obtaining assurance that a security policy is actually enforced by the Trusted Computing Base (TCB) of a system. As stated in the Criteria "the bounds of the TCB equate to the 'security perimeter' " and "includes all those portions .. essential to the support of the policy." That is, the security perimeter is with respect to the security policy being enforced. Thus, the two categories of policy, viz., mandatory and discretionary, may well have two distinct security perimeters. This, of course, only applies to systems of Class B1 or above, because Class C systems do not support a mandatory policy.

The mandatory policy, for both secrecy and integrity, can be enforced with a very high degree of assurance against concerted attacks, including Trojan horses. As the evaluation classes move from B1 to B2, B3, and finally A1, the primary distinctions relate to the use of improved architecture, specification, verification, and testing to increase the assurance in the mandatory access controls enforced by the TCB. It is expected that the higher evaluation classes will be used to protect against users with a wider range of authorizations.

In contrast, because of their richer policies, discretionary access controls have inherent limitations (known as the 'safety problem'[3]) and more complex mechanisms than mandatory controls. This is especially true for database systems that protect data at the granularity of individual elements and have powerful access mechanisms, such as views, which rely on much of the database system for their support. Because of the inherent as well as technological limitations, little meaningful assurance of discretionary controls can be obtained beyond that of Class C2; in particular, one cannot obtain high assurance against Trojan horses. Fortunately, this matches well the real-world need for discretionary controls for need-to-know and corresponding integrity enforcement. Moreover, because discretionary controls operate within the confines of mandatory controls, the damage that can result from their failure is limited.

Because of the sharp distinction in the possible assurance for mandatory versus discretionary controls in a database system, the following discussion presumes that there may be two distinct security perimeters for systems at Class B2 and above: an inner perimeter (the 'reference monitor') for mandatory controls and an outer perimeter (or perimeters) for discretionary and consistency controls. The maximum assurance that seems required, and the maximum practical, for the portion of the TCB outside the mandatory perimeter appears to be that prescribed for Class C2.

As discussed later, the assurance requirements for Class B2 and above, in particular the need to control covert channels, affects the meaning of consistency and the functionality of other aspects of a database system. However, having separate security perimeters makes it possible to more meaningfully address these problems.

AUTHORIZATION INTEGRITY

Mandatory Integrity Authorization

Mandatory security policies are particularly important because they describe global and persistent properties that are required for authorizations in a secure system. As defined in the Criteria[1], mandatory policies employ a reliable label to reflect the degree of protection required for information and to reflect the authorization of a subject to access information. When considering integrity, these labels reflect what the Criteria refers to as the 'sensitivity designation of the information,' or what is commonly termed the *integrity access class*, or simply *integrity class* of the information objects. There is a comparable label that reflects an individual's 'authorization for the information;' this label is assigned to corresponding subjects. The primary systems of interest are those that can be represented by a Formal Security Policy Model, as defined in the Criteria. For such a system it is shown that if the initial state of the system is secure with respect to the policy, then all future states of the system will be secure.

For mandatory secrecy policies, the secrecy access classes must form a lattice. This requirement may be appropriate for mandatory integrity policies as well, although nonlattice mandatory integrity policies have been proposed[4]. For lattice-based policies, the integrity classes could correspond to integrity levels (analogous to secrecy levels such as SECRET), category sets of disjoint integrity compartments (analogous to secrecy compartments such as CRYPTO), or both.

Six mandatory security policies have been variously proposed to deal with integrity. In the context of the above concept of mandatory policy, each of these is examined as a possible integrity policy for databases:

1. Strict integrity
2. Low-water mark
3. Ring policy
4. Multilevel security with no write-up
5. Program integrity
6. Domains and types

The first three policies were introduced by Biba[5] as possible policies for multilevel-secure systems.

Strict Integrity Policy. This policy is an exact dual of multilevel secrecy as defined in the Bell and LaPadula model[6]. Each subject and object is assigned a fixed integrity class taken from the lattice of integrity classes, and strict integrity is preserved by prohibiting a subject from reading down or writing up in integrity.

There are two distinct considerations in assigning integrity classes to objects and subjects. First, the integrity class of the object to be protected from unauthorized modification must reflect the sensitivity of the information, viz., the potential damage that could result. Second, the integrity class of the subject must reflect its trustworthiness for making modifications. However, it is essential to note that the modifications by a subject are effected by the programs it executes and the data that control the execution of these programs. Thus, if a high integrity class is assigned to objects (files or segments) containing programs and program data, this assignment must reflect a determination that the resulting execution will produce only acceptable modifications.

The strict integrity model was initially introduced to deal with the threat of deliberate falsification or contamination of very sensitive information. One such application in which high integrity is of great importance is the preparation of targeting data that are used to control ballistic missiles. The practical threat is not so much that an unauthorized individual will be allowed to use such a system, but rather that a program and/or data maliciously prepared will be incorporated into a Trojan horse to retarget the weapons towards inconsequential or even friendly targets. This kind of Trojan horse could be implanted in what has become popularly known as a 'virus,' and strict integrity has been recognized as one of the few effective defenses.

There is a growing body of experience with the implementation and use of strict integrity in highly trusted operating systems. For example, in the Honeywell SCOMP, the first Class A1 system on the Evaluated Products List, strict integrity is included as part of the protection for segments. This mechanism is used for the protection for security related information such as audit data. In addition, the Gemini GEMSOS[7] has incorporated strict integrity as part of the sensitivity label for all subjects, objects, and devices; this approach has been found useful when designing the integrity protection both of sensitive application information and of system information used to support the security controls themselves. Although there has been little comparable experience in database systems, the I.P. Sharp multilevel database model[8] incorporates strict integrity along with multilevel secrecy.

Low-Water Mark Policy. This policy is analogous to the high-water mark security policy of the ADEPT-50 system[9]. A subject's integrity class is dynamic and decreases as the subject reads data of lower integrity. If the integrity classes of objects are static (as in the strict integrity policy), a subject will be unable to write into an object with a higher integrity class than it has read; if the object classes are dynamic, then their integrity classes are possibly lowered if the subject writes into the object. As summarized by Biba[5], "This policy, in practice, has rather disagreeable behavior. . . . In a sense, a subject can sabotage (inadvertently) its own processing by making objects necessary for its function inaccessible (for modification). The problem is serious since there is no recovery short of reinitializing the subject." To the best of our knowledge, this policy has not been included in any system design.

Ring Policy. By prohibiting read-downs in integrity class, it seems the strict integrity policy and the low-water mark policy could prove to be quite restrictive for most systems, especially database systems. Because database processes must have both read and write access to user data, system tables, index files, logs, and other structures to answer queries and update the database, it would appear that the only workable assignment of integrity classes is system low. Because of the restrictiveness of the two preceding policies, Biba also introduced a more flexible policy called the ring policy. Each subject and object has a fixed integrity class, and a subject is only allowed to write into objects whose integrity classes are dominated by the subject's class. No restrictions are placed on reading, so a subject can write high integrity data even if it has read data of a lower integrity. Unfortunately, the relaxation of this policy makes the integrity class of the subject essentially meaningless, because there are no restrictions on even what programs the subject can execute. Thus, what would appear to be a high integrity subject can, without restriction, be executing erroneous or malicious programs that destroy the high integrity information to which the subject has access. In reality, this policy fails to meet the requirements for a mandatory policy. Moreover, there is no real experience using this policy as a basis for mandatory integrity.

Multilevel Security with No Write-Up. Extending the Bell and LaPadula model to prohibit 'writing-up' in secrecy class provides a limited form of mandatory integrity. In particular, this extended policy model addresses the 'write-up' problem of the mandatory secrecy policy, which allows a subject to write up in secrecy class. The extended model would prevent a SECRET subject, for example, from inserting data labeled as TOP-SECRET into a multilevel relation or from overwriting a TOP-SECRET element (which it cannot observe). This approach appears to protect subjects from lower-level subjects. Closer examination makes it clear that this approach is a case of the ring policy just addressed in which the secrecy labels, such as SECRET, are also used as the integrity labels; the difference is thus only syntactic with no difference in the results of the policy. Of course, this policy also has the same weaknesses as the ring policy.

Program Integrity Policy The restrictions of the strict integrity policy remain a concern, so it seems important to try to identify a more flexible but useful policy. The real world supports some notion of integrity class through job levels and chain of command. However, the flows between different levels (usually adjacent) are bidirectional, so information flows both up and down in integrity class. Moreover, the trust placed on the information provided by any individual is often more a function of the individual than position. The key to the effective protection in this context is that the individuals are trusted to make only the desired modifications of high integrity information, even though they have been exposed to information of lower integrity classes.

This same concept can be applied to software by imposing more stringent requirements on assigning an object containing executable code a high integrity class. It seems unreasonable to assume that once a program has observed data of low integrity that it is incapable of writing data of higher integrity, or because data are entered by a user of low integrity into a database, that indexes and other structures on the database must be treated of low integrity also -- there is little relationship between the quality of the data that go into a database and the quality of the system structures that represent it.

This problem has been approached by distinguishing read access from execute access (which are treated identically in the preceding policies). Based on this distinction, Shirley and Schell[10] have defined a program integrity policy in which a subject is only allowed to write into objects of less than or equal integrity class and only allowed to execute objects of greater than or equal integrity. As with the ring policy, there are no restrictions on reading. This policy appears to be better suited for databases because the database processes could operate with a high integrity class, where they would be able to read and update the entire database. Users and application processes would be assigned integrity classes reflecting their 'trustworthiness'. Furthermore, Shirley has shown not only that this is a mandatory policy but also that it is the identical policy implemented by the hardware protection ring mechanism of Multics and several other systems (no connection with Biba's use of the term 'ring'). Thus there is a substantial body of experience with this policy, and it has indeed been shown to be quite useful in operating systems. There is no comparable body of direct experience with database systems.

An even closer look at the program integrity policy reveals the somewhat unexpected result that it is just a special case of the strict integrity policy. To understand this, it should be recalled that in the Bell and LaPadula model there is the notion of a 'trusted subject.' When interpreted for integrity, as in the case of the strict integrity policy, a trusted subject is trusted exactly to be able to read low integrity information without damaging the integrity of high integrity data. This notion of trusted subject is too coarse for the problem at hand because a trusted subject can read any integrity class. However, the notion has been refined in the Gemini GEMSOS[7] to identify a 'multilevel subject' that has both a minimum and maximum class. Now, if the subject in each protection ring is regarded as multilevel (with respect to integrity classes) with a maximum integrity equal to the ring of execution and a minimum integrity equal to the least trusted ring, the strict integrity policy in this case becomes the program integrity policy if the multilevel subject is trusted not to execute any program with a lower integrity class than its maximum.

Domains and Types. Domains and types have been proposed as a means to specify a mandatory integrity policy, as illustrated by the Honeywell SAT system[4]. Here, each object is typed, and each domain has a list of types that it can observe and modify plus a list of domains that it can call. Although this policy model is similar to discretionary policies based on the access matrix model, the set of types, domains, and rights cannot be altered. Because it is a relatively new approach, its properties are not yet completely clear. So far, there is no experience applying this type of policy to a database system, although Honeywell is working on it.

Discretionary Integrity Authorization

Discretionary integrity authorization policies control access to data at the user or user group level. The usual approach to controlling access in database systems includes *authorization lists*, which specify what operations a user (or group) is authorized to perform on some set of data. For integrity, the operations of interest include update, insert, and delete.

The authorization lists of database systems are included in the data model at different layers of abstraction. At the lowest layer, they are associated with files, records, or elements. At the highest layer, they are associated with *views* or *subschema* on the data. The high-level approach has the advantage of specifying a context for access. The context -- i.e., exact set of elements that fall within the target of a view -- is dynamic, changing as the underlying database is updated. Because it is easier and more natural for users, the high-level approach has proven to be far more useful than the low-level approach, and is embodied in many systems including SQL/DS, DB2, ORACLE, and INGRES (though in a somewhat different form).

The discretionary security policy contained in the Trusted Computer System Evaluation Criteria[1] is appropriate for database systems as long as the concept of object is interpreted to mean views (actually view specifications or subschema) rather than just physical elements, records, or files. Note that this does not mean that discretionary controls cannot be associated with individual records and elements; such controls are easily defined as views on the database.

The Criteria specify that discretionary controls are to be applied to 'each named object.' There is no requirement that the named objects be disjoint in memory, and in some operating systems a file may be accessed via different path names through different directories with different discretionary authorizations placed on the different names. Similarly, applying discretionary controls to views is consistent with the Criteria because views are just a way of naming objects. Also, there is no requirement that the 'named objects' of the discretionary policy be the same objects or even at the same layer of abstraction as the 'storage objects' of the mandatory policy.

CONSISTENCY INTEGRITY

Database Integrity Rules

Database integrity rules protect a database from data entry errors as well as from other errors made by the operator or by software. They define the correct states of the database and may specify actions to take if an update would cause the database to enter an incorrect state. They are similar to exception conditions built into programs, except that the conditions are represented in the database (as metadata) rather than in the application programs so that they can be automatically applied to all transactions updating the database.

In a relational system, there are two common types of database integrity rules: domain integrity rules and relational integrity rules. *Domain integrity rules* are context-free rules specifying the allowable set of values (i.e., domain) for an attribute, e.g., DRIVER.AGE is greater than 16 but less than 100. *Relational integrity rules* are context-sensitive rules specifying more global constraints on individual tuples or sets of related tuples, e.g., that every tuple in a PROGRAMMER relation has a corresponding tuple in an EMPLOYEE relation (this is a form of 'referential integrity'). Many relational systems, e.g., INGRES, provide mechanisms whereby users can define rather complex integrity rules.

Integrity rules play a vital part in ensuring the integrity of a database. Indeed, they are a very important part of access controls because most systems are vulnerable to errors as well as to sabotage. It is probably fair to say that a database system would not be regarded as a useful trusted system if it does not support integrity rules.

There are, however, intrinsic problems associated with integrity rules in a multilevel system that is rated at the evaluation level of B2 or higher, arising from the requirement to protect against covert channels. Because the implementation of integrity rules is outside the mandatory security perimeter, the database subjects that enforce the integrity rules must be denied access to data that is classified higher than the subject level. Thus, if the subjects are processing a transaction on behalf of a user, the only data visible to those subjects will be data that is classified at a level dominated by the user's level. If the database system were given access to data not dominated by the user's level, then a Trojan Horse in the database system could leak the unauthorized data -- that is, unless the database system (or a large portion thereof) were part of the mandatory security perimeter. Because the latter is neither feasible nor desirable, in multilevel systems rated at the level of B2 or higher, we are forced to consider integrity constraints as constraints on the subset of the database dominated by the user's clearance.

To see how this revised interpretation of integrity constraints affects the enforcement of integrity rules, consider the relational model, which requires each tuple in a relation to have a unique primary key. Suppose the tuples in a multilevel relation are classified SECRET or TOP-SECRET, and suppose the relation contains a TOP-SECRET tuple with primary key FOO. This tuple will be invisible to subjects operating on behalf of SECRET users. Thus, if a SECRET user attempts to insert a new tuple, also with key FOO, the system will accept the tuple. Because the access class becomes the only means of distinguishing the tuples, the class must then be considered to be part of the primary key. We refer to the coexistence of multiple tuples with the same primary key except for access class as *polyinstantiated* tuples[11].

Problems also arise with respect to referential integrity. For example, suppose a TOP-SECRET user creates a TOP-SECRET tuple in a relation T(ID, A), which is associated with a SECRET tuple in a relation S(ID, B) through the join attribute ID. The relation S represents the entities named by the primary key ID. If a SECRET user deletes the referenced tuple in S, referential integrity will be violated. But because the SECRET user, as well as all subjects that run on that user's behalf, cannot know of the existence of the TOP-SECRET tuple, this cannot be avoided.

As a third example of the problems that arise from invisible data, consider a relation that contains the weights of items on board various flights. Suppose there is maximum weight restriction of 5000 for any given flight and that some of the items on board a flight are classified SECRET while others are TOP-SECRET. If the integrity constraint is specified simply as an upper bound of 5000 for the total of all weights for a flight, a flight could be overloaded because the TOP-SECRET weights would be invisible when the constraint is applied at the SECRET level to determine whether an additional SECRET item can be placed on board. A possible solution is to have separate constraints for SECRET and TOP-SECRET weights.

Thus, in B2 or higher systems, the consistency defined by integrity constraints must be interpreted with respect to the secrecy class of the subject applying the constraint. However, whether there should be some notion of inter-level consistency, or how this might be specified, is unclear. It is also unclear how triggers fit into this notion since a trigger activated by an operation on behalf of a user having one secrecy class cannot read up or write down in secrecy class. Finally, we note that if the database is polyinstantiated at the tuple or element level, problems arise in applying the integrity constraints because more than one tuple or element with different values may be selected by the constraint, each with different outcomes. Thus, the integrity rules must specify which values to select among polyinstantiated values.

In a multilevel system, the concept of integrity constraints should also be extended to include constraints on the classifications assigned to data. For relational systems, we have found that several properties should hold:

- The complete definition (schema) for a relation, including the names of all attributes, should have a single access class that is dominated by the access classes of all data that is to go into the relation. Integrity rules that constrain the data going into the relation should also be assigned this access class.

- The attributes representing the primary key in a relation should be uniformly classified -- that is, within any given tuple, the elements forming the primary key should have the same access class.

- The classification of the primary key should be dominated by the classifications of all other elements within a tuple.

In that integrity rules enforce constraints on the relationships among data in the database, they can be associated with inference problems. For example, if an integrity constraint states that $C = A + B$ for attributes A, B, and C, where A and B are SECRET but C is TOP-SECRET, then a SECRET user with access to A, B, and the integrity constraint can infer C. In this particular case, the best strategy for dealing with the problem may be to use the integrity constraint to force classifications on the data to prevent the inference -- e.g., classify A or B, or both, as TOP-SECRET. In cases where the rule of inference is complex and unknown, it may be more appropriate to classify the integrity constraint (which can be viewed as an inference rule).

In summary, although a multilevel secure database system should provide database integrity rules, the mandatory secrecy policy affects the interpretation and application of integrity constraints.

Recovery Management

Another vital aspect of database integrity is protecting the database from operator or software errors, including system crashes. The accepted method of dealing with such errors and faults is based on the concept of a *transaction*, which is a sequence of operations that behaves atomically -- that is, it either successfully completes (*commits*) all updates or else it has no effect on the state of the database (*rolls back*). The overall integrity policy for trusted systems should include the concept of transactions with commit and roll-back.

Multilevel updates raise some difficult issues regarding transaction management. For example, if a trusted user can simultaneously insert or update multilevel data (within the user's range of trust), it may be desirable to decompose these updates into single-level updates represented as single-level transactions and performed by single-level database subjects. However, the unit itself must also be treated as a transaction, so the concept of a multilevel transaction with single-level nested transactions appears to be very useful. The problem is rolling back the low portions of the transaction if the high portions fail.

Assuming recovery management is outside of the mandatory security perimeter, it is not clear how the database recovery log should be managed and processed in systems that are rated at the level of B2 or higher. However, some of the techniques used for general-purpose operating systems to ensure the consistency of file systems during backup and recovery may be useful.

Concurrency Controls

An important aspect of database integrity is ensuring that concurrent transactions do not interfere with each other, giving rise to inconsistent states of the database. *Serializability*, which states that any transaction schedule must be equivalent to one in which the transactions execute serially, has been shown to be a necessary and sufficient condition for global consistency[12], although there are systems that enforce somewhat weaker policies. Some notion of global consistency, however, is an essential aspect of the overall integrity policy for trusted database management systems. The concurrency policy should also address the problems of *deadlock*, where multiple transactions cannot proceed because they are waiting on each other, and *livelock*, where a transaction never exits from a wait state, both of which create denial-of-service problems.

In B2 or higher systems, the concurrency mechanisms must use techniques other than simple locks because read-write locks on multilevel data provide a signalling channel. Event counters[13] are not vulnerable to covert channels, but require that higher-level transactions roll back when a lower-level one causes an update that could interfere with its behavior.

CONCLUSIONS

We do not know enough about the application of mandatory integrity policies to databases to recommend any one in particular or even state that one be mandated at all. While the strict integrity policy without trusted subjects may be appropriate for some threat environments, the more flexible program integrity policy, which uses restricted trusted subjects to manage a database, may be appropriate for most environments. It would be premature to adapt a particular mandatory policy in criteria for trusted database systems until such a policy has been experimentally tried in at least one operational environment and has been demonstrably successful. On the other hand, a discretionary policy along the lines of that given in the criteria is extremely useful provided it is interpreted to apply to views rather than just elements, records, or files.

Database integrity rules should be included in an overall integrity policy because they provide users with considerable assurance that the data is protected against many errors. This is one of the best ways in which the users themselves can greatly enhance the integrity of their data. However, the interpretation and application of integrity rules is constrained by the requirements for mandatory security. Similarly, any trusted system should support the concepts of atomic transactions, recovery, and noninterference, though again the features are constrained by the mandatory security requirements.

Although we believe it is vital for trusted systems to support these different integrity policies, it is neither necessary nor possible to have the same degree of assurance in the enforcement of them all. Whereas Classes A and B are appropriate for mandatory access controls, Class C2 is appropriate for discretionary controls and consistency controls, which are considerably more complex than mandatory controls and require much of the database system for their support.

To provide a high degree of assurance, the mandatory integrity policy must be enforced by the reference monitor. In addition to enforcing the mandatory secrecy policy, the reference monitor ensures the integrity of all data in the system, including the labels that represent the secrecy and integrity access classes. If the data are vulnerable to tampering during storage or transmission to and from the reference monitor, cryptographic checksums may be used to ensure the integrity of the data and its labels. For cryptographic checksums to be meaningful, it is essential that the processes that compute and validate the checksums and manage the key be under the strict control of a reference monitor.

ACKNOWLEDGMENTS

An earlier version of this paper was prepared for the National Computer Security Center's Invitational Workshop on Database Security, where both authors participated in a working group on integrity and inference. The current version has benefited greatly from the group discussions, and we would like to thank the other group members, namely A. Arsenault, W. E. Boebert, D. Bonyun, D. Downs, K. Jacobs, R. Miller, G. Raudnbaugh, J. Spain, and S. Walker. We also thank T. Lunt, M. Heckman, and P. Neumann for their comments on this paper. This research was supported by the U.S. Air Force, RADC under contract F30602-85-C-0243.

REFERENCES

1. Dept. of Defense, Computer Security Center, *Department of Defense Trusted Computer System Evaluation Criteria*, 1983, CSC-STD-001-83

2. Date, C. J., *An Introduction to Database Systems*, Addison-Wesley, Vol. II, 1983.

3. Harrison, M. A., Ruzzo, W. L. and Ullman, J. D., "Protection in Operating Systems", *Comm. ACM*, Vol. 19, No. 8, Aug. 1976, pp. 461-471.

4. Boebert, W. E. and Kain, R. Y., "A Practical Alternative to Hierarchical Integrity Policies", *Proc. of the 8th DOD/NBS Computer Security Conf.*, 1985, pp. 18-27.

5. Biba, K. J., "Integrity Considerations for Secure Computer Systems", Tech. report ESD-TR-76-372, USAF Electronic Systems Division, Bedford, Mass., April 1977.

6. Bell, D. E. and LaPadula, L. J., "Secure Computer Systems: Mathematical Foundations and Model", Tech. report M74-244, The MITRE Corp., Bedford, Mass., May 1973.

7. Schell, R. R., Tao, T. F., and Heckman, M., "Designing the GEMSOS Security Kernel for Security and Performance", *Proc. 8th Dod/NBS Computer Security Conf.*, 1985, pp. 108-119.

8. Grohn, M. J., "A Model of a Protected Data Management System", Tech. report ESD-TR-76-289, I. P. Sharp Assoc. Ltd., June 1976.

9. Weissman, C., "Security Controls in the ADEPT-50 Time-Sharing System", *Proc. Fall Jt. Computer Conf.*, Vol. 351969, pp. 119-133.

10. Shirley, L. J. and Schell, R. R., "Mechanism Sufficiency Validation by Assignment", *Proc. of the 1981 Symp. on Security and Privacy*, Apr. 1981, pp. 26-32.

11. Lunt, T. F., Denning, D. E., Schell, R. R., Heckman, M., "Polyinstantiation in a Secure Relational Database System", Tech. report, SRI International, May 1986.

12. Rosenkrantz, D. J., Stearns, R. E., and Lewis, P. M., "Consistency and Serializability in Concurrent Database Systems", *SIAM J. Comp.*, Vol. 13, No. 3, Aug. 1984, pp. 508-530.

13. Reed, D. P. and Kanodia, R. K., "Synchronization with Eventcounts and Sequencers", *Comm. ACM*, Vol. 22, No. 2, Feb. 1979, pp. 115-123.

TRUSTED DATABASE DESIGN

Peter J. Troxell

PAR Government Systems Corporation
220 Seneca Turnpike
New Hartford, New York 13413

INTRODUCTION

In January of 1981, the Department of Defense Computer Security Center (DoDCSC) was formed to study all aspects of computer security and to promote the development of trusted computer systems. Their first task was to develop a set of criteria for defining what "trusted" meant, and for assigning levels to define how "trusted" a system is. Their first criteria, the "Department of Defense Trusted Computer System Evaluation Criteria [TCSEC]", was published in August of 1983. The "Department of Defense Trusted Network Evaluation Criteria [TNEC]", expected out in 1986, deals with network security issues.

This paper will discuss the software and hardware components which must be present in order for a Database Management System (DBMS) to be considered "trusted" in relation to the [TCSEC]. Distributed databases utilizing the TNEC will be considered beyond the scope of this paper.

Key Security Concepts

Several concepts must be addressed before any discussion of computer security can be made. The following paragraphs provide a general overview of these concepts so that later references to them may be understood.

The term "Trusted Computer System" is defined in the [TCSEC] as "a system that employs sufficient hardware and software integrity measures to allow its use for processing simultaneously a range of sensitive or classified information." In other words a user running at the Unclassified level can share the system with users running Top Secret, while ensuring that each user can access only those items for which they have permission.

The reference monitor concept developed from a study performed for the Air Force by James P. Anderson & Company. Simply stated, the concept stipulated that was that "a reference monitor which enforces the authorized access relationships between subjects and objects of a system" should exist. The mechanism that performs this concept is called a validation mechanism, and must meet the following three requirements:

a. "The reference validation mechanism must be tamper proof.

b. The reference validation mechanism must always be invoked.

c. The reference validation mechanism must be small enough to be subject to analysis and tests, the completeness of which can be assured."

This validation mechanism is given the name of the Trusted Computing Base (TCB) and is sometimes referred to as a security kernel.

The following excerpt from the mandatory security control policy defined in the [TCSEC] adequately defines the policy's meaning: "(the TCB) must include a set of rules for controlling access based directly on a comparison of the subject's clearance or authorization for the information and the classification or sensitivity designation of the information being sought, and indirectly on considerations of physical and other environmental factors of control."

Likewise, the control policy for discretionary security policy states that the TCB "must include a consistent set of rules for controlling and limiting access based on identified individuals who have been determined to have a need-to-know for the information."

TRUSTED COMPUTER SYSTEM EVALUATION CRITERIA

While this does not primarily address database security issues, it will be discussed since it presents some key concepts that are applicable to any multilevel secure software product.

Fundamental Security Requirements

The criteria presents six fundamental computer security requirements broken into three main categories of policy, accountability, and assurance. Each of these requirements is presented below with its rationale.

The first two requirements deal with policy. The first requirement states that there must be an explicit and well-defined security policy enforced by the system. As will be seen in the evaluation class, there are two types of policy -- mandatory for access rules to sensitive objects, and discretionary for allowing access by groups or individual users. For a mandatory security policy to work each object within the system must have an associated security label. This is the second requirement.

The third and fourth requirements focus on accountability factors. The idea is that each subject in the system will be identified and that security-related actions can be audited and traced back to the responsible party.

The last two requirements deal with assurance. This means that there is some way to guarantee that the first four requirements are enforced and that they are continuously protected against tampering and/or unauthorized changes.

Division Ratings

When a computer system is evaluated by the DoDCSC, it is assigned a rating. The rating consists of a division letter and a class number. The heirarchy of division and class numbers is as follows: the lower the division letter the higher the protection the system gives. As the class numbers increase within a division so does the security rating. Thus a rating of B1 is higher than a rating of C2 thus affording more protection. A key feature of the security ratings is, that inherited in the requirements for a particular class are all the requirements for any clasess lower than it in the hierarchy.

Division "D" contains only one class and is used only when a system that is evaluated does not fall in any of the higher classes.

All classes in division "C" implement some type of discretionary security policy. This will enforce a need-to-know type of protection on users and objects. Accountability is another feature and requires that certain audit capabilities be implemented.

A division "B" rating requires addition of a mandatory security policy. This policy requires sensitivity labels for all objects to be part of the major data structures of the system. Thus, the mandatory security policy supplementary to the discretionary policy developed for the division C systems. In addition, the system developer must provide the model of the security policy that the TCB is based on, along with its specification. The developer must also provide evidence that the reference monitor concept has been implemented.

For a system to receive a division "A" rating, it is required that the mandatory and discretionary security policies can be formally proven. The TCB is guaranteed that it meets its security requirements in all phases of design, development, and implementation. This guarantee is the result of adding formal methods into the design process.

TRUSTED DATABASE DESIGN

The need for a trusted DBMS arises from the fact that the [TCSEC] enforces access controls only to the granularity of a file. To make maximum use of a computer and its associated databases, these access controls must be expanded to arbitrate accesses to a finer detail, such as to the field or data element level.

The remainder of this paper will first discuss the security threats to a DBMS, then proceed to present some of the suggested approaches.

Security Threats

Two security threats, inference and aggregation, are prevalant in DBMS systems. In addition, there are those threats which can be found in any type of computer program, Trojan Horses and Covert Channels.

Inference, as the name implies, occurs when the user is able to infer some fact from the information that has been presented. Suppose, for example, that a database has two relations: AIRCRAFT, with attributes ID and PAYLOAD; and WEAPONS, with attribute TYPE and ID.The fields AIRCRAFT.PAYLOAD and WEAPONS.ID can be joined. All records are SECRET unless the WEAPON.TYPE is NUCLEAR in which case it is TOP SECRET. Now consider the following query:

```
RETRIEVE AIRCRAFT.ID
WHERE AIRCRAFT.PAYLOAD = WEAPON.ID
  AND WEAPON.TYPE = "NUCLEAR"
```

The query would be processed and would return to the SECRET user a list of all aircraft having a nuclear payload, thus revealing TOP SECRET information. This occurs because the computer treats the information returned as SECRET since the TOP SECRET portion was stripped away in the selection.

Aggregation occurs when data combined from different sources results in a data item that has a higher classification than its individual components. This can be the result of using one of the aggregate operations, such as sum, or can be intrepreted as the user, infering from different database requests, the data at a "higher"security level. For instance, in the previous example suppose that all records were SECRET but the fact that a particular aircraft was carrying a nuclear payload (i.e., the join relation) is TOP SECRET. By placing two queries a SECRET user could determine what the payload (TOP SECRET) was.

Other Security Threats A DBMS would, like its operating system counterpart, have to concern itself with the problems of Trojan Horses and Covert Channels. The [TCSEC] defines these two terms as:

Trojan Horse - "A computer program with an apparently or actually useful function that contains additional (hidden) functions that surreptitiously exploit legitimate authorizations of the invoking process to the detriment of security."

Covert Channel - "A communication channel that allows a process to transfer information in a manner that violates the system's security policy."

As can be seen from their definition care must be taken to prevent the occurence of these security threats.

Architectures

With the objective of having a Multilevel DBMS (MDBMS) and knowing the types of threats to the system, several potential architectures have been put before the community as potential solutions. These architectures are presented below.

It should be noted that whatever architecture is used, the concepts defined in the [TCSEC] will prevail. Each will contain, in some part, a TCB in which resides the security-related code that is guarenteed to work. Depending on the MDBMS, it will contain code to enforce mandatory and discretionary security policies. Marvin Schaefer [SCHA85] states in his paper that the [TCSEC] is sufficient in its current form to handle the multilevel database management problem, since each operating system maintains some type of internal database to keep track of its information.

Another key point is that all accesses to the database must be through the DBMS; otherwise, security is circumvented. This can be accomplished by making the database a special classification that can only be accessed from the DBMS which operates at that level.

Views The concept of views has been around since the early days of DBMS. In 1983, Billy Claybrook [CLAY83] presented a method for using views to enforce security requirements on a DBMS. A view is defined in [CLAY83] as "a database description (or definition), together with an instance of the definition." A view definition "is the process of specifing the attributes of a view and defining the mapping between the view and the underlying database."

The concept that a view utilizes is that a user is given access to a view but not the data itself so that the user will only be able to access what the view "sees." In addition, a view could be defined in terms of another view allowing for a breakdown of the component of the database tuples. The security classification can be either static or dynamic depending on to what depth the security labeling is taken.

A problem with this architecture its side effects due to the fact that the user only sees a part of the tuple For instance, if a user has delete permission to a tuple and subsequently deletes a tuple that was in the users view, what should be done with those attributes that were contained in the actual view but not "seen" by the user?

The [CLAY83] paper presents the author's method for handling the inference and aggregation problems. The solution to the inference threat was to make sure that the user had the necessary clearance for at least the highest object searched. Likewise, the solution presented for aggregation called for the user's clearance to match the highest classification in the material searched.

Integrity-Lock Richard Graubart has presented several papers ([GRAU84], [GRAU85]) on an architecture called Integrity-Lock. This architecture was an outgrowth of the 1982 Summer Study on Database Security sponsored by the Air Force Studies Board. Its key architectural concept is to be able to retrofit security onto existing DBMS instead of recreating the DBMS from scratch.

The Integrity-Lock approach calls for the database management system to be separated into three components. Graubart's conception of this is shown in Figure 1. The trusted code resides in the Trusted Front End (TFE). The TFE is responsible for authenticating the user, and verifying that only information that the user is entitled to, is passed back to him. The Untrusted Front End (UFTE) takes care of parsing the queries and formatting the output for the user. Lastly, the Untrusted DBMS handles all the I/O access to the actual database.

The [GRAU84] paper goes on to define the basic theme of the Integrity-Lock architecture; that is each tuple has at least one classification attribute and an associated cryptographic checksum. This provides a mechanism for labeling the classification of the data and provides a way to detect unauthorized modifications to the tuple. The checksum is computed using the value of the tuple and its classification as input to an "unbreakable" encryption algorithm. The result is placed with the tuple in the database. Should an unauthorized modification be made to the data, the checksum will not match and a security violation flagged. Dorothy Denning, in her 1984 paper [DENN84], presents just how these checksums can be computed along with their strengths and weaknesses.

The granularity of the security level can be increased anywhere from the tuple level to individual attributes by the addition of classification/checksum pairs. Of course the greater the granularity, the greater the cost; in terms of CPU power to compute the checksums, and the amount of disk space require to save the database.

One of the key advantages of this architecture is that the technology needed to implement it currently exists. It can be retrofitted on to an existing DBMS to reduce the cost and time required to have a Multilevel DBMS in the marketplace.

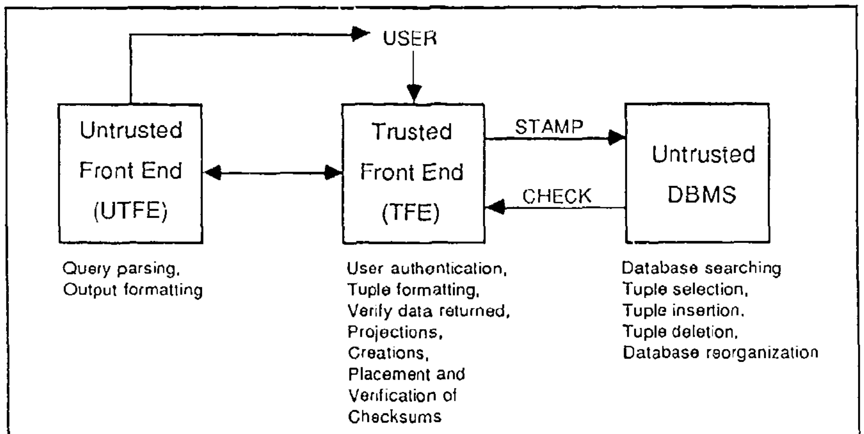

USER
Untrusted Front End (UTFE)
Trusted Front End (TFE)
STAMP
CHECK
Untrusted DBMS
Query parsing, Output formatting
User authentication, Tuple formatting, Verify data returned, Projections, Creations, Placement and Verification of Checksums
Database searching Tuple selection, Tuple insertion, Tuple deletion, Database reorganization

FIGURE 1 - INTEGRITY LOCK COMPONENTS

CONCLUSION

This paper has presented an overview on the development of "trusted databases." It has discussed the threats to such a database and has presented a brief overview of some of the current ideas for a likely architecture. While each has its own strengths and weaknesses, time will tell which, if either, will be the final solution.

These designs have dealt with the implementation of the mandatory security policy onto an existing DBMS. Further work still needs to be done on how to implement the discretionary security policy of need-to-know onto a database, be it either at the database level or at the level of individual attributes.

REFERENCES

[CLAY83] Claybrook, B. G., "Using Views in a Multilevel Secure Database Management System," *Proceedings of the 1983 Symposium on Security and Privacy*, IEEE Computer Society, 1983, pp 4-17.

[DENN84] Denning, D. E., "Cryptographic Checksums for Multilevel Database Security," *Proceedings of the 1984 Symposium on Security and Privacy*, IEEE Computer Society, 1984, pp 52-61.

[DENN85] Denning, D. E., "Commutative Filters for Reducing Inference Threats in Multilevel Database Systems," *Proceedings of the 1985 Symposium on Security and Privacy*, IEEE Computer Society, 1985, pp 134-146.

[FERR81] Fernandez, E. B., Summers, R. C., Wood, C., Database Security and Integrity, Addison-Wesley Publishing Company, Inc., Reading, Massachusetts, 1981.

[GRAU84] Graubart, R. E., "The Integrity-Lock Approach to Secure Database Management," *Proceedings of the 1984 Symposium on Security and Privacy*, IEEE Computer Society, 1984, pp 62-74.

[GRAU85] Graubart, R. E. and Duffy, K. J., "Design Overview for Retrofitting Integrity-Lock Architecture onto a Commercial DBMS," *Proceedings of the 1985 Symposium on Security and Privacy*, IEEE Computer Society, 1985, pp 147-159.

[RADC75] Hinke, T. H. and Schaefer, M. "Secure Data Management System," RADC-TR-75-266, Rome Air Developement Center, Air Force Systems Command, Griffiss Air Force Base, New York, November 1975.

[SCHA85] Schaefer, M., "On the Logical Extension of the Criteria Principles to the Design of Multilevel Database Management Systems", *Proc. of the 8th National Computer Security Conference*, DoD Computer Security Center, 1985, pp 28-30.

[TCSEC] Department of Defense Trusted Computer System Evaluation Criteria. Department of Defense, CSC-STD-001-83, 15 August 1983.

[TNEC] Department of Defense Trusted Network Evaluation Criteria. Department of Defense, DRAFT, 29 July 1985.

THE CHALLENGE AFTER A1
A VIEW OF THE SECURITY MARKET

Lester J. Fraim

Honeywell Information Systems
Federal Systems Division
7900 Westpark Drive
McLean, Virginia 22102

INTRODUCTION

Honeywell Information Systems has the only two commercial products on the National Computer Security Center's (NCSC) Evaluated Products List above class C2. The Multics Product is rated as a class B2 and the Scomp is the only system to receive the highest rating of class A1. These systems are used in a variety of applications where security is a key requirement. Several new developments are underway to further demonstrate the effective use of the Scomp to meet a variety of market needs.

As a result of the experience with Multics and Scomp, Honeywell is developing a strategy and product direction to expand our segment of the evolving security market. The security market consists of several elements which must be integrated into a coordinated set of product and service offerings.

This paper will present a view of the security market and discuss the initial approach being taken to develop products to meet these market needs.

BACKGROUND

Honeywell has long been committed to the development of systems to meet the security needs of government and industry. The development in the early 1980's are key examples of this effort. Bringing trusted products to the marketplace has provided Honeywell with a unique view of security market requirements. The advent of the Trusted Computer System Evaluation Criteria and overall awareness of trusted system concepts has grown rapidly during this period. The list of vendors now working with the NCSC is a relative who's who in the industry. Each vendor must decide the position (i.e., rating) and type of products to be offered. The end result will be that all products will contain enriched security mechanisms. Vendors will provide standard products which meet a broad spectrum of security and processing requirements.

As a leader in the class B2 to class A1 area of trusted products, Honeywell has developed a basic strategy to meet the needs of this market. The overall strategy includes the coordination of security related efforts through Honeywell. A high level steering group reviews plans and requirements to ensure that the technical security efforts are directed with a unified goal in mind. This group meets regularly to evaluate product characteristics, program results, market requirements and research directions. The direction provided by this group ensures that the efforts of various organizations are all directed to meet the security needs of the Honeywell customers.

Key to Honeywell's effort is the inclusion of enhanced security in both our large and small commercial product bases. Without basic products which meet the evolving standards, it is quite unlikely that we can provide complete solutions to the high end of the market. Another key element in the strategy is to ensure that research is programmed into product enhancements. One example of this is the inclusion of Scomp hardware features in the DPS6 PLUS product, which was announced in June this year. This commercial hardware platform contains the features to support the Scomp capability. This will enable the evolution of enriched security features to be implemented in the commercial operating system offering as well as provide a new platform for Scomp. The Secure Ada Target (SAT) Program is also being managed such that this technology can be planned for product offerings at the proper time.

As the technology evolves, Honeywell will insert product offerings which take advantage of proven technology. This approach, however, can produce some difficult challenges. With each new innovation comes the need to define the security impacts, implementation approach and the application of the technology. These are the challenges that make the trusted system market interesting. The Scomp system was a major technical accomplishment because it demonstrated the ability to build a class A1 system. The challenge now is to build a broad product offering, meeting high level security requirements and providing all the features available in the non-trusted market. To understand these requirements will require a quick look at the characteristics of this marketplace.

MARKET CHARACTERISTICS

The Honeywell experience with Scomp and Multics has given us the opportunity to evaluate a variety of system requirements. Because these systems are very different with respect to capacity, performance and capability, we have observed requirements across a broad spectrum. This experience has led us to the definition of a marketplace model. This model looks very similar to many system and program requirements. It is not much different from the model of the data processing industry in general. Technology has provided the capability to place large processing capacity at user locations and provide effective communications between these processing elements.

A key element of the market, which is not obvious from the model, is the need for solutions oriented systems. These systems must solve the users problem and provide the level of trust necessary for the user environment. There is no attempt here to justify the users security requirements. Security must be an element of the specifications just as communications interfaces and processing requirements. The market requires systems which solve user problems and protect their processing assets.

The Market Model

Figure 1 illustrates the interconnections of several classes of processing elements. The elements are interconnected through a Local Area Network (LAN). There are efforts underway by several vendors to produce trusted LAN products. This model does not depend on their capability; however, these products will enhance the vendor's ability to satisfy the model requirements. The LAN is required to provide efficient control between the processing elements. The elements span the spectrum of what is available in the market today. The challenge is that all elements need to be trusted at the class B2 to class A1 level.

To establish a common understanding of these elements, it is necessary to describe some of their features.

Trusted Work Stations - The requirement for trusted work stations is quite straight forward. Users desire the full capability of work stations, including color graphics, windowing, disk storage, a mouse and hard copy capability. Work stations run a variety of software including MS-DOS* and UNIX.** The challenge is to provide these features, meet the security requirements, and allow all applications software to run without modification.

Trusted Servers - These are departmental size systems which provide a broad range of processing resources. These systems manage the data resource for the users. This data management may be in the form of a relational data base management system, document management system, or file management capability. This system manages the data resource for the user, and enforces the security policy.

* MS is a trademark of Microsoft.

** UNIX is a trademark of AT&T Bell Laboratories.

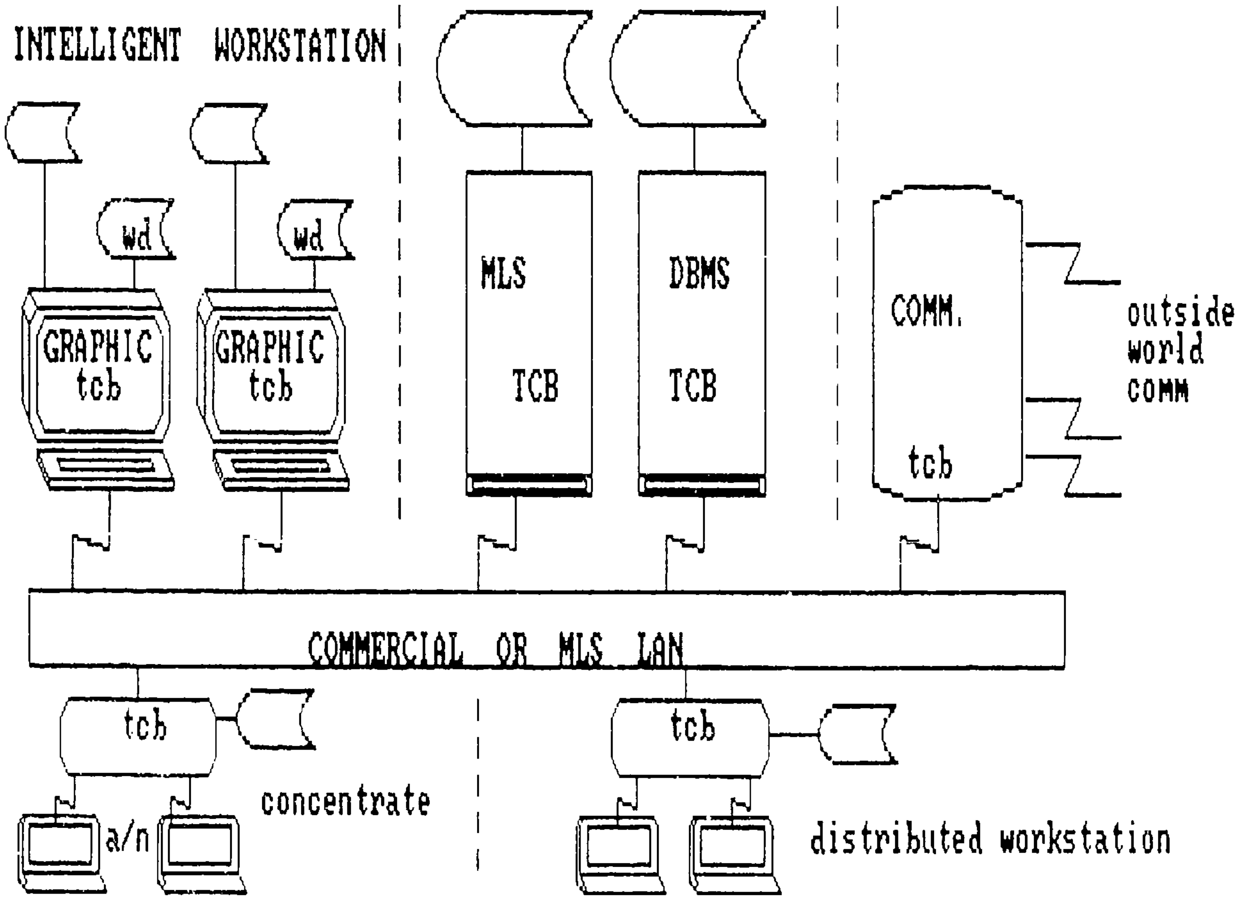


Figure 1. Trusted System Market Model.

Trusted Gateways - This element of the model provides access to the outside world. This function allows users to access information from external sources. Many requirements exist to protect a local resource (i.e., LAN) from unauthorized access. The Gateway provides this protection, and also allows system users to gain access to other non-local environments. In the Government, these gateways will require TCP/IP capabilities. In the commercial world, the gateways will probably require ISO or SNA capabilities.

Trusted LAN Access - The development of trusted LANs may preclude the need for this element of the model; however, the functions are still required. The trusted LAN access element will ensure the separation of levels on the LAN, and provide a trusted interface to the LAN mechanism. The concept is to provide an effective user interface to the LAN.

Standards - This market model is driven by standards. Everyone involved with system requirements is quite familiar with both official standards and the evolving standards of practice. For example, there are several standards for LAN connections. Most notably are those of the IEEE. These standards are very different from the evolutionary standards of practice such as UNIX System V for departmental processing and UNIX or MS-DOS for work stations. To meet the requirements of the market model, the vendor must identify the standards to be supported and the standards of practice which will be supported.

Application - One of the major lessons learned with the Scomp product has to do with applications requirements. Everyone wants to see applications running which perform functions for the user. The challenge with applications comes from several sources.

First, there are commodity applications which users would like to use. These include such things as data base management, spreadsheet, word processing, transaction processing, etc. So the first challenge is to be able to support a variety of these existing applications in the trusted environment.

Second, there are many applications which require a security model which is different from that supported by the basic trusted system. Examples of these applications include guards, military message system and data base management. These applications require trusted elements which cannot just be ported from commodity packages. The challenge is to develop effective trusted interfaces which meet a wide variety of market requirements.

And finally, there are applications which must be trusted because they are required to handle multi-level objects. An example of this kind of application would be interfaces to networks that contain multiple level traffic. None of these exist today, with the possible exception of AUTODIN. To meet this challenge will require applications such as trusted TCP/IP or X.25 capabilities.

Solutions - To meet the needs of the market place requires a strong combination of product and integration capabilities. The products will provide a foundation for the building of system solution. The vendor must be committed to long term investment to bring the technology and solutions to the customer. To meet the needs of the market, The vendor will have to combine the traditional vendor role with the system integration role. The key to success in this arena is the commitment to meeting the customer requirements. These solution oriented systems will all require elements of the model, and each may include a unique piece that is only an emerging technology. A strong technically oriented organization will be the most successful in meeting these solution needs.

THE FIRST STEP

As can be seen from this quick review of the trusted market requirements, there is a great deal of work ahead. There are also many new and interesting challenges in bringing these capabilities to the market. At Honeywell, we have been working to take the initial steps to begin to address the various elements of the market model. By no means do we have solutions for all the elements or all the issues. That is the challenge to this industry over the next several decades.

We plan on building on the technology of the Scomp product by producing systems which meet the requirements of the model. These systems will then be used in our solution oriented business to meet customer requirements. As new technology is advanced, it will also be integrated into solutions. As other vendors provide elements necessary to meet our users' needs, we will integrate them into sound technical solutions.

Because of the nature of Scomp, and the type of system it provides, our initial capability will be in the departmental sized system. It is well known that Scomp currently resides on a 16 bit mini-computer hardware platform. This hardware is modified to meet the needs of building a trusted system. Several years ago steps were taken to ensure that these hardware characteristics would be available in the future Honeywell hardware platform. This was accomplished through close working relationships between the commercial hardware developers and the Scomp development team. The results of this effort are the newly announced DPS 6 PLUS product set. This commercial product provides a long term technically advanced base for the Scomp system. Additionally, the Scomp hardware features ensure that future versions of the commercial operating system will be able to provide enhanced security capabilities. It is now planned that the future commercial operating system will be targeted at class B2.

DPS6 PLUS

The DPS6 PLUS is a new generation of 32-bit virtual memory computers. It is built using Very Large Scale Integrated (VLSI) chips as integral elements of the central processor and the memory manager. The central processor firmware is loaded via the System Management Facility to control software operation.

The major significance of the DPS6 PLUS is that it provides a commercial hardware platform, without the need for special hardware to support the Scomp Trusted Operating Program (STOP). The initialization of the system will be achieved through the firmware load mechanism of the System Management Facility. This feature of the DPS6 PLUS will provide a great deal of flexibility and a reduction in product cost.

Figure 2 lists several of the features of the DPS6 PLUS and the Scomp. As can be seen, the DPS6 PLUS provides a multi-processor capability with a large virtual address per process. The performance of the system is enhanced by the integration of the Scientific and Commercial Instruction Processors (CIP/SIP). These processors were not available on the 16 bit Scomp implementation. Additionally, the largest segment size and the availability of twice as many segments performance.

Because of the firmware load capability, the DPS6 PLUS implementation of Scomp will be able to use commercial Test and Verification (T&V) routines. The current Scomp requires a unique set of T&V's because of the hardware differences. This will be a major cost savings in the DPS6 PLUS based product.

The firmware load capability is also beneficial in providing the mechanisms necessary to implement the one Scomp feature not in the DPS6 PLUS hardware. The commercial DPS6 PLUS only provides support for physical Input/Output (IO). Firmware will be added which supports the virtual IO capabilities necessary for Scomp. Because of this difference, only pre-mapped IO will be supported on the DPS6 PLUS. The mapped IO feature of the current Scomp will not be available.

STOP 3.0

The first version of the Scomp operating system to be available on the DPS6 PLUS is defined as STOP 3.0. This is the same operating system which runs on the 16 bit Scomp except that it is modified to support the features of the DPS6 PLUS. These modifications include the larger segments, multiple processors, new IO capability, and new ring crossing mechanisms. The user interface to STOP will remain the same, and the application interface to the system will be the same.

Multiple Processor Support - The 16 bit Scomp was implemented on a mono processor system. The DPS6 PLUS supports single, dual and quad processor configurations. The Scomp Kernel is being redesigned to effectively support the multiple processor environment. This is a complex enhancement to the Scomp security Kernel, and has taken the most effort to design.

New IO Support - The current Scomp supports user initiated IO capabilities. This will no longer be true on the DPS6 PLUS implementation. The IO capability will be moved into a more privileged ring (ring 1), and the system will perform the IO on behalf of the user. This change is required because of the following.

First, the IO environment on the DPS6 PLUS is quite different from that on Scomp. There is no firmware support for some existing functions. Secondly, the development of new smart device controllers requires the IO mechanism to be protected from the user environment.

	SCOMP	DPS6 PLUS
WORD SIZE	16 Bit	32 Bit
SECURITY FEATURES	Add-On-Hardware	Commercial Hardware
VIRTUAL ADDRESS SPACE	2 Mega Bytes	2 Gigabytes
SEGMENTS/PROCESS	512	1024
SEGMENT SIZE	4 Kilobytes	2 Megabytes
PAGE SIZE	256 Bytes	2 Kilobytes
SIP/CIP SUPPORT	None	CPU - Supported
MULTI-PROCESSOR	Mono	Mono-Dual-Quad
PHYSICAL ADDRESS	2 Megabytes	16 Megabytes
FIRMWARE	ROM	RAM

Figure 2. DPS6 PLUS/SCOMP Features.

Application

The initial application to be supported on the DPS6 PLUS Scomp will be the UNIPLEX II* integrated office application. The implementation of UNIPLEX II is currently being completed on the 16-bit Scomp System. UNIPLEX II consists of word processing, file management, data base management, spreas sheet and mail. UNIPLEX II normally runs on UNIX based systems, and has been ported to Scomp using our evolving C standard application environment.

Evaluation Goals

The DPS6 PLUS with STOP 3.0 will be built in accordance with the class A1 requirements. The system's initial evaluation goal will be a class B3 rating. The reason for this is that it reduces the risk associated with verification. There are many issues associated with this product enhancement that need to be addressed in the verification aspects of class A1. An example of this is multiple processors and smart controllers. Additionally, the technology of verification has not advanced significantly from the current Scomp product to warrant major investment in this technology at this time. Future versions of STOP, however, will be validated at the class A1 level when it is required to meer the market requirements. Nothing will be done in the development of STOP 3.0 which would preclude it from achieving the class A1 rating.

Performance

The performance range of the DPS6 PLUS extends the performance capability of the SCOMP system. There will be additional performance benefits gained from the larger segment size and the integrated SIP/CIP capability. Looking at the performance of the DPS6 PLUS and the DPS6/75 produces the following results:

o SCOMP (DPS6/75)	=	1.0
o PS6 PLUS 1	=	1.7
o DPS6 PLUS 2	=	3.2
o DPS6 PLUS 4	=	5.2

These performance ratios are based on the basic hardware, and have not been factored to reflect the impact of security.

THE FUTURE

STOP 3.0 is just the first of a planned evolution of systems capability on the DPS6 PLUS platform. Additional interfaces and applications will be developed to meet the needs of the market model. These efforts will come from both internal and project directed funding. Several of these additions are being planned at this time. They include a relational data base management capability, DDN capability, Ethernet** interface and additional support tools.

* UNIPLEX is a trademark of Redwood Int. Ltd.
** Ethernet is a registered trademark of Xerox Corporation.

Application Environment

The market is driving the departmental system toward the UNIX System V interface as a standard. In line with this, the STOP application interface is being enhanced to make the porting of applications from UNIX as easy as possible. This is the result of several efforts to port UNIX based applications to Scomp. These applications include TCP/IP, X.25, UNIPLEX II and a C-Compiler.

This is not an effort to emulate the UNIX environment. It is purely a mapping of UNIX interface calls to services provided by STOP. Our approach is to provide a trusted system which supports UNIX applications. Not all applications will port easily.

Data Base Management - It is not possible to provide true MLS relational data base capabilities today. However, the use of a commercial RDBMS capability on a trusted system is the first step toward realizing many of the requirements of a RDBMS in a trusted environment. Honeywell plans on addressing this need by providing basic data management capabilities on future versions of Scomp. The approach to meet this requirement has not been fully defined. Work is continuing in several areas to address the data base management requirements.

Other Products - The DPS6 PLUS Scomp is only one element of the market model. Efforts are underway at Honeywell to address the full spectrum of the model requirements. These are being addressed both in terms of product capabilities and as evolving research issues. The use of the DPS6 PLUS chip set is being analyzed with respect to development of a micro based capability which could meet the needs of a work station or communications device. These efforts are in their early stages and should produce meaningful results in the next several years.

Additionally, a key research activity being performed by the Honeywell Secure Computing Technology Center (SCTC) is being monitored for inclusion in product oriented solutions. The Secure Ada Target (SAT) research provides a potential path to advanced security mechanisms. The timed inclusion of the proven technology developed by SCTC will be a key element in the development of advanced products.

CONCLUSION

This paper has looked at the requirements of the Trusted System market place. These requirements cannot all be met with existing product platforms and capabilities. This market requires a strong solution oriented approach combined with basic platforms to meet the users security needs.

Honeywell has come a long way in achieving the Scomp and Multics evaluations. These efforts, however, are only preliminary to our eventual goal of providing a broad range of product oriented solutions. The DPS6 PLUS is the key element of this evolutionary approach to trusted product development. The DPS6 PLUS, combined with standard interfaces and applications environments, will provide a set of quality solutions for systems users.

SE/VMS: IMPLEMENTING MANDATORY SECURITY IN VAX/VMS

Steven Blotcky, Kevin Lynch, Steven Lipner
Digital Equipment Corporation
Nashua, NH and Littleton, MA

ABSTRACT

Since the late seventies, Digital Equipment Corporation has been pursuing a development program aimed at improving the security of its computer system and network products. The most visible product of this program to date has been Version 4.2 of the VAX/VMS operating system, which is under evaluation as a candidate for Class C2 of the Trusted Computer System Evaluation Criteria. In addition to implementing discretionary access controls, VAX/VMS Version 4.2 incorporates latent support for mandatory security controls at the level of internal operating system routines and data structures. This paper describes SE/VMS (Security Enhanced VMS), a set of modifications that allow VAX/VMS users to exploit the latent support for mandatory security. The modifications provide facilities that allow a system manager to set up and administer the mandatory security environment, and that allow users to operate on labeled objects. The paper describes the functions of SE/VMS that support user registration and login, device and volume management, file creation and access, and the production of labeled printed output. Discussions are provided of the techniques that were used to implement SE/VMS, of the system's limitations, and of plans to gain user experience with SE/VMS. SE/VMS is viewed as providing an interim mandatory security capability for VAX/VMS users, and will not be submitted for evaluation at Class B1 of the Criteria.

1 INTRODUCTION

Since the late seventies, Digital Equipment Corporation has been pursuing an active development program aimed at improving the security of our computer system and network products. The primary focus of this program has been a series of enhancements to the security of the VAX/VMS operating system. The most visible product of the program to date has been VAX/VMS Version 4.2, which has been submitted for evaluation at Class C2 of the Trusted Computer System Evaluation Criteria[1] (TCSEC).

This paper describes SE/VMS (Security Enhanced VMS), modifications that have been developed to provide an initial mandatory security capability for VAX/VMS. These modifications were developed by Digital's Software Services organization to provide labeled security protection for VAX/VMS. This work is intended to meet most of the requirements for Class B1 of the TCSEC. Because SE/VMS does not meet all requirements and is intended to provide only an interim capability, it would not be a candidate for submission for formal product evaluation at Class B1.

SE/VMS is not an "add-on" security package in the sense of some of the products on the National Computer Security Center's Evaluated Products List. Instead it combines latent capabilities of VAX/VMS, replacements for some VAX/VMS components, and additional components to achieve the overall objective of providing labeled protection.

This paper begins with a review of the security features of VAX/VMS Version 4.2. It then summarizes the support for mandatory security that was included in Version 4.2. Next, the paper presents an overview of the features of SE/VMS along with a sketch of the techniques that were used to implement them. Finally, we conclude with a discussion of areas for future development in providing mandatory security for VAX/VMS.

2 SECURITY IN VAX/VMS

VAX/VMS was initially developed in the mid seventies along with the VAX-11/780 32-bit superminicomputer. The VAX-11/780 was developed as an upward-compatible extension to the PDP-11 minicomputer family and executes PDP-11 code directly. As the VAX family grew out of the PDP-11, so VAX/VMS grew out of the RSX-11/M operating system for the PDP-11.

Initial releases of VAX/VMS actually included a significant number of PDP-11 utility programs that were transported unmodified from RSX. Thus the initial VAX/VMS security design was an extended "minicomputer" model and encompassed passwords at login and "system/owner/group/world" protection on files, directories and a few other objects. VAX/VMS has always supported one-way encryption of user passwords, and over the years a number of security auditing functions were incorporated with the system's accounting features.

In the late seventies and early eighties, a major project was started with the aim of upgrading the security of VAX/VMS. The first product of this project was VAX/VMS

Version 4.0, and some additional enhancements were incorporated in Version 4.2. When this paper discusses the features of SE/VMS, it describes changes or enhancements to Version 4.2. Because the initial implementation of mandatory controls was incorporated in Version 4.0, the paragraphs below will refer to Version 4.0 in some cases. (Odd-numbered versions since 4.0 have been dedicated to "bug fixes" rather than significant feature enhancements.) As it currently exists, VAX/VMS Version 4.2 incorporates the following security enhancements:

- o A number of "account management" features including account expiration, restrictions on days and times of login, and restrictions on access to accounts (no dialup, no network, etc.).
- o A number of password management features including required change of initial passwords for privileged accounts, password expiration, minimum password length, dual-password accounts, and a random pronounceable password generator.
- o Features directed toward systems that support dialup lines or networks including automatic hangup and limits on unsuccessful login attempts directed to an account.
- o Access control list and identifier features allow the system manager to define arbitrary groups of users, and allow users to grant or deny access to files by individual users or defined groups.
- o Selective security auditing features produce an audit trial of successful and/or failed attempts at such operations as user login, access to files, and use of certain privileges. The audit trail is directed both to a terminal and a log file, and can be analyzed by a reduction procedure included in the system.
- o Features introduced in VAX/VMS Version 4.0 prevent "disk scavenging" by insuring that disk files are erased on deletion, or that blocks newly allocated to files are pre-erased. VAX/VMS systems have always erased primary memory pages before making them addressable to a process, so the enhancement to disk storage allocation eliminates the last possibility for disclosure of information by object reuse.
- o A "secure server" key prevents users from implementing "password grabbers" by guaranteeing that a user of a hardwired terminal who presses the break key will always receive a login prompt from the operating system. Equivalent features are provided for users whose terminals are attached to terminal concentrators or VAX network hosts.
- o A "Guide to VAX/VMS System Security"[2] was developed along with VAX/VMS Version 4.0, and updated for Version 4.2. The guide provides detailed information for both users and system managers.

The development of VAX/VMS Version 4.0 was started before the completion of the final version of the TCSEC. Nonetheless, the developers were aware of the Criteria development process, and tracked the content of each draft of the TCSEC. A specific goal of VAX/VMS Version 4.2 was that it meet the requirements of Class C2, Controlled Access Protection. VAX/VMS Version 4.2 has been under formal evaluation[3] as a candidate for Class C2 since late 1985.

3 MANDATORY CONTROLS FOR VAX/VMS

While the primary security evaluation goal for VAX/VMS Version 4.0 was to meet the requirements of Class C2 of the TCSEC, it was understood during the development process that incorporation of mandatory security controls was both a feasible and desirable objective. Resource limitations and time-to-market constraints prevented the completion of the mandatory security features. However, a good deal of work was completed, and "latent support" for mandatory security has been present in every release of VAX/VMS since Version 4.0.

Early in the development of VAX/VMS Version 4.0, a decision was made that the system would support both the lattice security and integrity[4] models, with fields allocated to support 256 levels and 64 categories for each of the security and integrity models. The fields were encoded in a conventional way - a byte each for security and integrity levels, and a 64-bit quadword for security and integrity category masks. These fields, plus an additional 16-bit word used as a filler, form a five longword structure known as an "access classification block", or CLS block. Thus, the total storage required to represent a security "access class" (levels and categories for security and integrity) is 160 bits. As part of the development of VAX/VMS Version 4.0, CLS blocks were added to the data structures for the system's subjects and objects.

The security properties of a subject are recorded in a CLS block within an "Agent's Rights Block", or ARB, that includes the subject's current access class as well as identity, group and privilege information that is used for the other protection checks performed by Version 4.0. The only subjects on a VMS system are processes.

The security properties of most objects (files, "mailboxes", logical name tables, devices, and global sections) that are active (accessible or "opened") in the system are stored in "Object's Rights Blocks" or ORBs. An ORB contains two CLS blocks, specifying minimum and maximum access classes for the object, as well as discretionary access control information. Other objects (e.g. mounted disk volumes) have CLS blocks as part of their control structure. While the major storage objects are labeled with CLS blocks, a few (less critical) interprocess communication objects are not labeled.

The ORB and ARB are data structures that apply to active subjects and objects in a VAX/VMS system -- processes that are logged in (ARB), and open files, logical name tables, and so on (ORB's). For mandatory security controls to be effective they must also, of course, apply to permanent subjects and objects - registered users, files, directories and volumes. Thus the system's permanent data structures were enhanced to record access class information. The User Authorization File (UAF) entry for a user records that user's minimum and maximum access class. The "volume home block" for a disk volume records the minimum and maximum access class for the volume, while the "file header" for each file records the file's access class. In all cases the standard VAX/VMS 160-bit CLS block is used to store the access class.

Volumes and devices may be multilevel (minimum and maximum access class may differ for each object, as set by the system manager) while a file always has a single access class. Directories are files with special properties and also have a single access class. Additional process control and communication objects (i.e. logical name tables, global sections, "mailboxes") are potentially multilevel objects.

In addition to adding access class information for subjects and objects, the VAX/VMS Version 4.0 development project also completed the code required to implement mandatory controls for files, and extended the executive's central protection checking routine to reflect the access class of subject and object in its decision to grant or deny access. Access checks and propagation of access classes were based directly on the requirements of the Bell-LaPadula model[5]. A subject may only read an object if the subject's access class dominates the object's access class (simple security condition). A subject may only write an object if the object's access class dominates the subject's access class (*-property or confinement property).

While the code that checks access was part of VAX/VMS Version 4.0, no provision was made to allow a subject to have a non-zero access class. Only in the case of files was a subject's access class propagated to objects it created as required by the Bell-LaPadula model's rules for creation of objects. Thus, there was no operational ability to label objects, only a latent one.

A pair of privileges -- downgrade and upgrade -- may be granted to a process to exempt it from the security and integrity *-properties respectively. The execution of the mandatory security access check in VAX/VMS Version 4.0 is conditioned on a global "sysgen" parameter: when the parameter is 1, checking is enabled. The sense of the encoding of access classes is such that, as long as the entire access class is zero, access is always granted. Thus a user who sets the sysgen parameter inadvertently will lose some processor time to access checks but will not find his system "broken".

The implementation of mandatory controls in VAX/VMS Version 4.0 provides a relatively complete set of structures and support in the operating system kernel for labeled security protection. However, no user (or system manager) interface to the mandatory access controls is provided, access class is only propagated for files, and mandatory access checks are not made during some operations (e.g. mounting disks). In addition, even though file access failures caused by a violation of mandatory security will appear in the system's audit trail, the reason for such failures (i.e. the incompatible access classes) will not.

If an installation is to make use of the mandatory security support in VAX/VMS, it must have a way to associate character-string names with levels and categories, to assign "clearances" to users, to allow users to select an access class at login, and to display access class information on printed output, in directory listings, and so on. In addition, a system manager must have facilities to set up a system, for example defining the access class ranges of drives, volumes, and terminals, and must have access to access class-related information in the system's audit trail.

A number of Digital's users have "discovered" the mandatory security features in VAX/VMS and written their own software to exploit them[6]. The experience of these users seems to show both the viability of the implementation of mandatory security controls in VAX/VMS Version 4.2 and the critical need of some users for these features.

4 SUPPORTING MANDATORY SECURITY IN VAX/VMS

This section describes the features and implementation of SE/VMS. In the following paragraphs, emphasis has been placed on the SE/VMS features that support mandatory security controls. As was mentioned above, integrity labeling is also present and supported in SE/VMS, but most mention of the integrity model has been omitted from the paragraphs below in an attempt to shorten and simplify the presentation.

4.1 Objectives

The discussion above has described the support for mandatory security controls that is present in VAX/VMS Version 4.2, as well as the support that has not yet been completed. The objective of the SE/VMS development was to provide near-term support for mandatory security. The ground rule of the development effort was to provide a complete and usable system, but to defer where necessary support for features or facilities that would unduly complicate or delay the provision of basic support. Specifically, it was decided not to modify any of the existing system data structures. No effort was made to add mandatory controls to any object that did not already have a CLS block in its associated data structures.

4.2 Approach

The technical approach to the development of SE/VMS was, as might be expected, to build on the support for mandatory security in VAX/VMS Version 4.2, and to add those components that were missing or incomplete in Version 4.2. In practice, this effort required a few changes to the basic Version 4.2 executive, the replacement of some Version 4.2 modules with enhanced ones, and the development of some entirely new modules. Because the VMS development group enhanced the latent support for mandatory security that had been present in Version 4.2 by adding system service routines to the executive for VAX/VMS Version 4.4, it was then decided that SE/VMS would be developed as a set of enhancements to Version 4.4.

The following sections describe the features that were added by SE/VMS and the general approaches to implementing those features. An overview of the implementation of SE/VMS is provided at the end of this section.

4.3 Names For Access Classes

VAX/VMS stores an access class (in a CLS block) as a purely numeric value. Therefore a mapping between the alphanumeric name of a security or integrity level or category and the corresponding encoded value is needed both for input (user registration, login, etc.) and output (directory listing, printed output).

The VAX/VMS rights database supports mapping between numeric values and alphanumeric identifiers (names) as part of the user group identifier mechanism mentioned above. A range of binary identifier values was reserved to hold the names of security and integrity levels and categories. A simple arithmetic conversion allows the VMS executive to transform the value corresponding to a level or the bit position corresponding to a category into a binary identifier value. Pre-existing mechanisms for processing the rights database implement the mapping between identifier value and alphanumeric name. VAX/VMS already provides a utility to maintain the rights database, as well as the User Authorization File (Authorize); commands were added to this utility that allow the system manager to specify the names of security and integrity levels and categories.

4.4 System Service Support

A uniform syntax was developed for the specification of access classes by users (Figure 1). This syntax allowed for the specification of classification information by an alphanumeric string (as described above), or by numeric value. The VMS development group provided two new system services in Version 4.4, one to parse ASCII access class strings and translate them into binary CLS blocks and a second to create an ASCII access class string from a CLS block.

```
(LEVEL=SECRET)
(CATEGORY=27)
(LEVEL=TOP SECRET,
    CATEGORY=(BLUE,RED))
(LEVEL=(MINIMUM:SECRET,
    MAXIMUM:TOP SECRET), CATEGORY=RED)
(LEVEL=(MINIMUM:UNCLASSIFIED,
    MAXIMUM:255), CATEGORIES=(1,3))
```

Figure 1. Examples of Valid Access Class Strings

A third system service was provided to set and get the access classes of those objects that have associated ORBs. These are the services that became available with VAX/VMS Version 4.4, and motivated the decision to implement SE/VMS under that version rather than Version 4.2.

4.5 Authorizing Users

The system manager who wishes to add a user to an SE/VMS system must be able to specify a "clearance" for that user. The VAX/VMS Authorize utility is normally used to register users and specify their security attributes. Authorize was modified for SE/VMS to accept user access class information. A syntax for entering such information was devised that is consistent with normal usage in VAX/VMS and Authorize (Figure 2). Because VAX/VMS already uses the "/SECURITY" command qualifier for other purposes, "/SECRECY" is used to specify the mandatory security clearance property.

```
UAF>ADD MODEEN/SECRECY=
     (LEVEL:(MINIMUM:UNCLASSIFIED,
      MAXIMUM:TOP SECRET),
      CATEGORY:(MAXIMUM:(APPLE,BANANA)))
```

Figure 2. Specifying User Clearance

A user can be allowed a single classification, or a range of classifications.

4.6 Logging In

The VAX/VMS LOGINOUT utility was modified to assign an access class to the user's process, and to validate that access class. When a user logs in interactively, an access class for his or her process can be specified using the standard syntax (Figure 3). If none is specified, the process will default to the user's maximum authorized access class.

```
     USERNAME: LIPNER/SEC=(LEVEL:SECRET,
                   CATEGORY:(BANANA,GRAPE))
```

Figure 3. Login With Classification Specified

The LOGINOUT utility then validates that the access class is between the user's minimum and maximum (as well as validating the login against the other information in the UAF). It also validates the requested access class for the login against the range of access classes authorized for the terminal (See below). LOGINOUT then stores the access class in the process' ARB. In the case of a non-interactive login, such as a submitted batch job, the process is assigned the user's maximum access class and validation is performed against the command, error and log files specified by the user.

4.7 Volumes And Devices

The system manager of a SE/VMS system will normally wish to specify the ranges of access classes for mass storage devices and volumes and for user terminals. A new command and associated utility program allow the system manager to specify the necessary parameters for objects with ORBs (Figure 4).

```
    SET CLASS/OBJECT_TYPE=DEVICE/SECRECY=
          (LEVEL:(MINIMUM:SECRET,
           MAXIMUM:TOP_SECRET),
           CATEGORY:(MAXIMUM:(APPLE,BANANA)))
                     DUA1:
```

Figure 4. Setting Device Access Class.

New switches (/SECRECY and /INTEGRITY) have been added to the INITIALIZE command (Figure 5) to allow a volume to be initialized so that only files within a specified range of access classes can be written to it. The INITIALIZE command operates on a disk volume that is physically mounted on the VAX system but not yet logically accessible to application programs. The access class is stored in the home block of the disk.

```
    INITIALIZE/SECRECY=(LEVEL:(MINIMUM:SECRET,
          MAXIMUM:TOP_SECRET)) USERDISK02
```

Figure 5. Setting Volume Access Class.

The SET CLASS commands may only be used by the system manager or a privileged user to change the classification of objects owned by the system. Their effect is to set the minimum and maximum access class values in the ORB for the specified object. Because the ORB is a transitory data structure, these commands must be repeated each time the system is rebooted. They will normally be included in a command procedure that is executed at system startup time before users may log in. This use of a command procedure is consistent with normal VAX/VMS practice.

When files on a volume are to be made accessible to SE/VMS users and programs, an option of the the SE/VMS MOUNT command compares the access class ranges of device and volume and, if the range of the volume is "within" that for the device, allows the mount to proceed. In this case, the MOUNT command copies the access class range for the volume into the device's ORB, saving the old device access class information so that it may be restored when the volume is dismounted. The MOUNT and SET CLASS commands allow the system manager to mount a foreign disk or tape volume at the access class of the device where the volume is to be mounted.

4.8 Operations On Files And Directories

As was mentioned in the discussion of mandatory controls in Version 4.0, the operations of object creation and initial access (file open) built into VAX/VMS implement the requirements of the Bell-LaPadula model in a straightforward fashion. A newly created file or directory inherits the access class of the creating process. Opens for reading and writing are subject to the constraints of the simple security condition and *-property.

As with any system that implements the lattice model and a hierarchical file system, SE/VMS enforces a "compatible" hierarchy in which the security classes of files and directories are monotonically non-decreasing (and integrity classes non-increasing) as one proceeds away from a volume's root directory. Any user can create an "upgraded" directory via the SET CLASS command, but will then be unable to gain access to the new directory without logging in at a higher access class. The files within a given directory will normally be at a uniform access class and only directories will be upgraded.

Any user who owns or uses files at multiple access classes will require a way to discover what files and directories are present at various access classes. The VMS DIRECTORY/FULL and DIRECTORY/SECURITY commands (requiring read access to the directory) have been modified for SE/VMS to produce a listing of file and directory names and access classes for user review.

The VAX/VMS BACKUP utility was modified to preserve the classifications of files and directories when they are backed up to tape or disk. Access checks are made during both backup and restore operations.

4.9 Additional Objects

Because of the structure of VAX/VMS, any object that has an associated ORB will be protected by the system's mandatory controls.

Logical name tables (used to translate names used by programs and the VAX/VMS command language), global sections (used to map files into shareable areas of main memory), and "mailboxes" (used for interprocess communication like Unix(tm) pipes) have associated ORB's and thus are protected by the system's mandatory controls.

These additional objects are created dynamically by processes in execution. The VMS executive was modified to set the access class of a newly created object of any of these types to the access class of the creating process, except in the case of a global section "backed" by a disk file; in that case the global section is given the access class of the file. The access classes of objects of these types may be altered by the SET CLASS command (given sufficient user privilege) and displayed by the corresponding SHOW CLASS command.

4.10 Labeling Output

For many users, the "bottom line" of a system that implements mandatory controls is the ability to produce properly labeled printed output. As part of the SE/VMS development, a print symbiont was developed that verifies the requesting user's mandatory access to a file, then produces a listing with labeled header and trailer pages and optional top and bottom labels on each page. The layout of the header, trailer, top and bottom labels are customizeable. A SE/VMS utility allows the format to be defined for each unique combination of security level and categories.

4.11 Auditing

The VAX/VMS security auditing facilities seemed to audit the "right things" for SE/VMS, but were insensitive to mandatory security access classes. For SE/VMS, the existing facilities were enhanced to record access class information where appropriate (login, file access).

To allow a reasonable level of audit selectivity at audit trail collection time and avoid flooding the system's audit log file, the VAX/VMS executive was modified to allow system manager selection of auditing of all file access at or above a selected security class. A command, SAUDIT, was implemented as part of SE/VMS to allow a system manager to select the access class threshold for auditing (Figure 6).

```
SAUDIT/ENABLE/SECRECY=(LEVEL:SECRET,
      CATEGORIES:(APPLE,GRAPE))
```

Figure 6. Selecting the Audit Threshold Access Class

4.12 Mail

The VAX/VMS MAIL utility is used to send messages between users. As distributed with Version 4.2, it would only be possible to send mail between users at the unclassified level. The SE/VMS development project modified MAIL so that a message can be sent from a process to any user who could read a file at the sending process' access class. In some cases, the receiver's copy of the message may have its access class raised to the receiver's minimum access class. The receiving process can only respond with a message built into the mail program that says "user HAS READ YOUR MESSAGE".

4.13 Implementation Considerations

The implementation of SE/VMS was simplified by the level of support for mandatory security already present in Versions 4.2 and 4.4 of VAX/VMS, and by the structure of VAX/VMS. The normal functions of an operating system kernel are performed by the VAX/VMS executive. The executive performs such functions as opening files and checking access. Support functions are performed by programs (images) that are part of the operating system, but run in the context of the process that invokes them. In some cases, these operating system images may have privileges of their own; more often they inherit any special privileges of the user on whose behalf they operate.

SE/VMS implements mandatory security controls in VAX/VMS by first enabling the mandatory control support features that are always present in the VAX/VMS executive. In a few cases, the executive has been modified (patched) to add features not yet supported by VAX/VMS. For example, selective auditing by security access class, and filling in ORBs with classification information are implemented by patches to the executive.

A number of the user and system manager support functions in SE/VMS are implemented by images that are present, but do not support mandatory controls, in the standard VMS product. In these cases, SE/VMS simply modifies the source programs for the images, then replaces these images at SE/VMS installation time. This is the case for the Authorize, LOGINOUT, and Directory utilities. In each case, the required modifications are localized to small segments of the image in question.

Finally, some of the components of SE/VMS required the development of entirely new programs (though perhaps based on existing VAX/VMS software). For example, the labeling print symbiont of SE/VMS and the SAUDIT command are in this category. In this case, too, SE/VMS simply installs the new program in a directory where it will be available to the system manager.

5 LIMITATIONS, EXPERIENCE AND FUTURE DIRECTIONS

5.1 Limitations And Support

The sections above should have made clear the fact that SE/VMS is intended to provide an initial mandatory control facility for VAX/VMS. This section considers what is "not provided" with SE/VMS.

The combination of VAX/VMS Version 4.4 with SE/VMS provides a fairly complete set of mandatory control facilities at the operating system level. Users' processes can create, delete, read, and write objects at the operating system level, and those operations will be constrained by and consistent with the requirements of the mandatory security controls.

Two major system objects - event flag clusters and lock blocks - are not labeled. Event flag clusters are sets of 32 bits, normally used for posting events, that can be used for interprocess communications. A process can access two shared event flag clusters at a time. Lock blocks are structures used to control access to shared resources. They can optionally be associated with a 16-byte value block that can be used to communicate information among processes sharing the resource. Both lock blocks and and event flag clusters are allocated dynamically by the system.

There are a few feature shortfalls that might be expected to be resolved in a full-fledged system. For example:

- o Terminals associated with terminal servers (such as DECserver-100s) can not be assigned access classes individually; all such terminals must be given the same access class as a group.
- o Some of the auditing facilities are relatively coarse and not well-tuned for the mandatory controls. For example, one cannot tell from the error coding in the audit trail whether a file access attempt was rejected because of the mandatory controls or the discretionary controls.

These and other equivalent shortcomings demonstrate that SE/VMS is still an evolving system at the operating system level, rather than a completely finished one.

The area where SE/VMS will present the greatest challenge to its users is not in the domain of operating system features, but in application structure. It is clear that an ordinary unprivileged VAX/VMS application program that does not attempt to cross access class boundaries will function correctly under SE/VMS. It is equally clear that a complex application that operates on multiple files, perhaps of different access classes, may find itself broken by SE/VMS.

Some complex applications must be installed "with privilege" in a VAX/VMS system. Those applications may have sufficient power to defeat SE/VMS, eliminating part of the benefit of the mandatory controls. On the other hand, some privileged applications (MAIL is an example) may not have enough power to overcome the mandatory controls. The key point is that there is a significant amount of engineering required to make complex applications operate correctly in an environment where mandatory security controls are being enforced, and that engineering has not yet been done for the applications that may be asked to operate under SE/VMS.

SE/VMS may interact in unexpected ways with VAX/VMS applications. A pool of specialists has been trained in mandatory controls in general and in SE/VMS in particular so they might understand their effects on applications. Such training can provide specialists with the skills necessary to provide support for mandatory controls in the future. This support, in addition to basic installation of the SE/VMS software, could include defining initial security policy, setting up device and directory structures, and analyzing the impact of SE/VMS on applications.

On hearing a description of the features of SE/VMS, a listener might naturally be expected to ask "has it been submitted for evaluation?" Digital believes that SE/VMS meets many of the TCSEC requirements for Class B1, Labeled Security Protection. However, absent a full developmental evaluation, it seems likely that there are specific features that fall short of the requirements of Class B1. In addition, the documentation for SE/VMS is not structured in accordance with the requirements of the TCSEC, and the requirements for complete functional testing of the security features have not been met. Digital has requested that NCSC initiate a developmental evaluation of SE/VMS. The intention of requesting this evaluation is primarily to provide better insight into what might be required to make a future release of VAX/VMS meet the requirements of Class B1.

5.2 Experience With SE/VMS

As part of its evaluation of the impact of mandatory controls on VMS and its users, Digital has provided copies of SE/VMS to a selected set of VAX/VMS users. Because this paper was prepared shortly after the evaluation copies of SE/VMS were distributed, there is no experience to report. It is anticipated that some comments on user experience with SE/VMS will be included in the presentation of the paper at the Ninth National Computer Security Conference.

5.3 Directions For The Future

The discussion above clearly points the way toward a possible future release of VAX/VMS meeting the TCSEC requirements for

Class B1. In addition, Digital is continuing advanced development projects aimed at evaluating the feasibility of developing a Class A1 security kernel that would be compatible witn VAX/VMS. Advanced development and architecture studies are also continuing to examine the impact of mandatory controls on VAX/VMS layered software products. An additional focus of advanced development work is the need for enhanced security in Digital's DECnet wide-area network and Ethernet local-area network products. As these advanced development projects reach maturity, they are likely to form the basis for future papers like this one.

REFERENCES

1. Department of Defense Trusted Computer System Evaluation Criteria, CSC-STD-ØØ1-83, Department of Defense Computer Security Center, Fort George G. Meade, MD 2Ø755, August 1983
2. Guide to VAX/VMS System Security, AA-Y51ØA-TE, AA-Y51ØA-T1, Digital Equipment Corp., Maynard, MA Ø1754, July 1985
3. Product Evaluation Bulletin, VAX/VMS Operating System, Version 4.2, Report Number CSC-PB-Ø1-85, National Computer Security Center, Fort George G. Meade, MD 2Ø755, October 1985
4. Biba, K.J., Integrity Considerations for Secure Computer Systems, ESD-TR-76-372, Electronic Systems Division, AFSC, Hanscom AFB, MA, April 1977
5. Bell, D.E. and LaPadula, L.J., Secure Computer Systems: Unified Exposition and Multics Interpretation, MTR-2997, MITRE Corp., Bedford, MA, March 1976
6. Technical Description of the VAX/VMS Version 4 Non-Discretionary Security Implementation, SAIC Comsystems, Chesapeake, Virginia, 1985

CAVEATS

This paper presents the opinions of its authors, which are not necessarily those of Digital Equipment Corporation. Opinions expressed in this paper must not be construed to imply any product commitment on the part of Digital Equipment Corporation.

The following are trademarks of the Digital Equipment Corporation: DEC, DECnet, DIGITAL, PDP, RSX, VAX, VMS.

Unix is a trademark of AT&T Bell Laboratories.

A VERIFIED LABELER FOR THE SECURE ADA TARGET

William D. Young*
Paul A. Telega
W. Earl Boebert
Honeywell Secure Computing Technology Center
St. Anthony, Minnesota

Richard Y. Kain
Department of Electrical Engineering
The University of Minnesota

Abstract: This paper describes the specification and verification of a prototype line printer labeler for the Secure Ada Target (SAT) machine currently under development at the Honeywell Secure Computing Technology Center. There are two types of constraints on a secure labeler--functionality requirements on the labeler itself, and constraints on the context in which the labeler is called. The approach described addresses both types of constraints. Verifying properties of the labeler itself is an interesting but straightforward exercise in program verification--in this case, code level verification. This verification alone, however, does not ensure that the labeler is unavoidably encountered in moving text from user domain to line printer or that the output of the labeler cannot be altered by user programs. Such constraints require the construction of an *assured pipeline* and are easily handled by the SAT type enforcement mechanism. Type enforcement is described and shown to have broad applicability in handling such context constraints.

INTRODUCTION

Designers of secure computing systems go to considerable lengths to guarantee the proper segregation of internal information. This care can be wasted if the information is compromised externally or at the I/O interface between the computer and its external environment. Thus, the DoD Trusted Computer Systems Evaluation Criteria[1] (TCSEC) specifies a labeling requirement on systems at or above the B level of certification. For human-readable output this requires that:

> The TCB [Trusted Computer Base] shall mark the beginning and end of all human-readable, paged, hardcopy output (e.g., line printer output) with human-readable sensitivity labels that properly represent the sensitivity of the output. The TCB shall, by default, mark the top and bottom of each page of ... output with human-readable sensitivity levels that properly represent the overall sensitivity of the output or that properly represent the sensitivity of the information on the page.

This paper describes one approach to satisfying this requirement--a prototype line printer labeler for the Secure Ada Target (SAT) machine currently under development at the Honeywell Secure Computing Technology Center. SAT is intended to satisfy or exceed all of the TCSEC requirements for A1 certification. Among these is the requirement for design verification. Consequently, the labeler described here has been designed so that it can be formally verified. This places constraints on the labeler that make the design somewhat less flexible than has apparently been true for most related efforts[2, 3]. We examine the implications of the requirement for formal verification on trusted software Labeling is one of a number of areas which require code which is commonly called *trusted*. However, unlike some other trusted software such as a downgrader[4], we invest trust in the code not because it is privileged to violate some aspect of the security policy but because its functioning is crucial to the maintenance of security in the system. For a discussion of this distinction see[5].

Our presentation is as follows: in section 2 we outline the security requirements for a labeler in an A1 context. Section 3 describes the SAT prototype line printer labeler and the way in which the security constraints have been met. Finally, we draw some conclusions in section 4.

*Also with the Institute for Computing Science and Computer Applications, The University of Texas at Austin

THE LABELING REQUIREMENTS

The basic requirement for a labeler is simply to associate the correct sensitivity label with a document and to guarantee that the label is affixed in such a way that it will appear in the proper format and position on the resulting human-readable output. This seems a simple requirement; for a line printer, for example, simply partition the input stream into a sequence of pages with an appropriate character string (the label) inserted at appropriate points in the output stream. Thus, the labeler procedure takes as input a character sequence and a security level (or the corresponding human-readable label associated with that level), and generates as output a character sequence with labels and page breaks inserted at the appropriate positions in the sequence.

However, the labeler is merely one program executing in concert with many others. Any assurance provided by the labeling process is lost if the input can be manipulated to insert, for example, top secret information into an input stream the labeler is to mark as unclassified. Similarly, the labeling requirement is circumvented if the output stream can be altered to replace any label by some string representing a label for a lower security level. Thus, there are two components to the labeling requirements: *correctness* constraints on the functionality of the labeler itself, and *integrity* constraints on the handling of documents in the "information pipeline" that ends at the line printer physical device

The correctness constraints are specifications on the labeler code. There is considerable flexibility in defining these constraints. For example, the TCSEC does not specify the output page format other than the placement of the labels; nor does it specify the particular characters permitted in the output sequence. To restrict the possibilities for covert channels in the output formatting[6] and because of the desire to formally verify the code, the SAT prototype labeler specification imposes fairly stringent restrictions on the labeler functionality. These may be stated as follows:

A1. The labeler must partition the input stream into pages, each of which begins and ends with a label. Pages are defined by the placement of carriage control characters in the output. This label must be the human-readable character string associated by the system administrator with the level of the document represented in the input stream. (We are considering only single-level documents in this discussion. Handling of multi-level objects and documents in the SAT is currently under consideration.) The page size and page width are device-dependent parameters. Output pages must satisfy these size constraints and contain only characters from a certain limited set.

A2 The document represented by the input stream must not be unacceptably altered by the labeling process. Acceptable alterations include the insertion of labels at the appropriate places, breaking lines that exceed the permitted line length, removing characters that are not within the permitted character set, and deleting characters that would be overstruck (The current design does not permit underlining or highlighting of text by overstriking. This limitation is a consequence of the way in which lines are maintained in the pagination process; the limitation could be changed in subsequent designs.) The sequence of printing characters in the output is a subsequence of the printing characters in the input.

The constraints on the environment in which the labeler is invoked are designed to preserve the integrity of its inputs and outputs. This is more a function of the overall security mechanism in SAT than of the labeler itself. These constraints may be stated as follows:

B1. The level associated with the (input) document must be an accurate representation of the sensitivity of the information contained in the document. This implies that the level of a document is not accessible to manipulation by arbitrary user programs. Moreover, the content of the document is not subject to alteration by arbitrary user programs.

B2. A stronger restriction is necessary to avoid mislabeling: the labeled document may only be output on a device for which it was labeled; no other manipulation should be possible. The output document must not be accessible to manipulation by arbitrary user programs.

B3. The labeler is limited to dealing with the files passed as parameters. That is, the labeler is constrained from accessing arbitrary files even if the system's general object access constraints (e.g., the mandatory and discretionary security policies) would otherwise allow it.

These restrictions on the handling of the document outside the labeler are more difficult to insure in most systems than the constraints on the behavior of the labeler itself. Verifying properties of the labeler merely involves examining the code of the labeler. The other properties relate to the environment in which the labeler is invoked. They reflect on the way in which general documents may be handled in the system. Systems that enforce the Bell and LaPadula model of security[7], for example, typically guarantee adherence to constraint B1. Initial assignment of levels is not a function of the system, but once information has been classified, the Simple Security Property and the *-Property ensure that high level information cannot flow into objects at lower levels. The Tranquility property requires that the level of an object remain fixed throughout its lifetime.

These mandatory constraints, however, do not prevent the manipulation of the labeler's output by user programs at appropriate levels. That is, a program operating on behalf of a top secret user might be able to alter the labels on a top secret output document without running afoul of the mandatory constraints. One might encapsulate the labeler and printer mechanisms so that there is no point at which intervention is possible. This encapsulation violates the principle of modular design that dictates that separate functions should reside in separate modules. Alternatively, one can impose additional integrity constraints which makes the output file inaccessible to user programs because their integrity level is too low; this is the SCOMP approach[5]. Boebert and Kain[8] have shown that hierarchical integrity approaches that are sufficient to meet the B1, B2, and B3 restrictions necessarily involve *trust* since data must "flow up" in integrity. The SAT *type enforcement* mechanism addresses these issues with a novel approach that subsumes hierarchical integrity policies[8, 9].

Very little work has been done on the labeling problem. Kurth [3] describes a line printer labeling package for an IBM/370-compatible machine with the MVS operating system. This differs from our work in that it describes a mechanism used in a single-level system, and is not formally verified. Rudell[2] examines the labeling of screen output at a fairly high granularity. Again, the system is not formally verified. The only verified routines similar in spirit to the SAT prototype labeler are the proofs of the trusted device-driver routines of SCOMP[10]. However, this verification was done at a very high level, and it was assumed that a process existed which did the labeling correctly. We are not aware of any implementation-level proof of a labeler process.

LABELING AND SAT

The treatment of labeling in the SAT system is presented in two parts. We first examine the labeler itself and the properties that it is proven to satisfy. These are constraints A1 and A2 of the previous section. We then present the SAT type enforcement mechanism and show how this preserves the integrity of the output data after it has been labeled (constraint B2). This mechanism is quite general, and we indicate how our particular problem is only a special instance of a more general problem of restricting access to classes of objects.

The Prototype Labeler

The functional correctness of the labeler is defined in terms of constraints A1 and A2 above. The input to the labeler is a sequence of characters and a level. The output is a sequence of characters that is the properly massaged version of the input--labels have been inserted at the appropriate places and the output sequence is a legitimate transformation of the input. The labeler was fully specified and mechanically verified using the Gypsy Verification Environment[11, 12]. The complete Gypsy text is given in the appendix.

For purposes of verification, the labeling process is broken into two steps. In step one, a paginator process breaks the input into a sequence of pages of correct size. Extraneous characters are discarded at this point. The paginator is verified to two properties: that the resulting sequence contains only correct pages, and that the printing characters in this sequence are a subset of those in the input. It is still conceivable that the labeler could signal information by the sequence of characters deleted. We consider this possibility unlikely and don't attempt to prevent it.

There are two global constants, Logical_Page_Length and Logical_Page_Width, in the specification that characterize the amount of space on a line printer page (minus the amount needed to add the labels at the top and bottom of the page). A correct page has exactly Logical_Page_Length lines, each of which is a sequence of at most Logical_Page_Width printing characters. Printing characters are those in the ASCII character set between space and "~", a range that excludes all control characters. (This range was chosen because certain devices allow device characteristics to be reset by sequences of control characters. Passing to the device sequences of characters which might reset page boundaries or selectively disable the print head might vitiate the labelling requirement. We simply disallow all control characters; a more selective filter is obviously desirable.) Other characters allowed in the paginator output are carriage return (CR), line feed (LF), and form feed (FF); these have their typical meaning in the division of the input into lines and pages. A FF in the input sequence, for example, causes the current page to be filled out with null lines and a new page to begin. Other ASCII characters in the input sequence are discarded. The formal (Gypsy) specification for pagination of the output is given by the following three recursive function definitions:

```
function CORRECT_PAGE_SEQUENCE (pages. pageseq): boolean =
begin
   exit (assume result iff
           (  pages = null(pageseq)
            or
              (  correct_page (first (pages))
               & correct_page_sequence (nonfirst (pages)))));
end: {correct_page_sequence}

function CORRECT_PAGE (pg  page): boolean =
begin
   exit (assume result iff
           (  size(pg) = logical_page_length
            & (all i: integer,
                  i in [1    size(pg)]
                -> correct_line (pg[i])))).
end. {correct_page}

function CORRECT_LINE (ln. line). boolean =
begin
   exit (assume result iff
           (  size(ln) le logical_page_width
            & (all i  integer.
                  i in [1    size(ln)]
                -> printing_character (ln[i])))).
end.  correct_line}
```

This specification may be considered slightly flawed in that references to *printing_character* should not appear in the formatting constraint, but rather in the textual integrity constraint described below. That is, for a line to be correct from a formatting standpoint it need only be of the correct length. Subsequent versions will include this change.

The other crucial property of the labeler is that it not distort the input. This is handled in a very simple fashion. As the input sequence is scanned, the printing characters are extracted. Those are the characters that are placed into the output pages. They are also recorded in a character sequence *purgetxt*, which is compared to the input sequence. The property that is proven is that the sequence of printing characters in *purgetxt* is equal to the printing characters of the input sequence extracted by a call to the function Purge_Text defined as follows:

```
function PURGE_TEXT (inseq  text). text =
begin
   exit (assume result =
            (if inseq = null(text)
                then null(text)
                else
                   (if printing_character (inseq[1])
                       then    inseq[1]
                               > purge_text (nonfirst (inseq))
                       else purge_text (nonfirst (inseq))
                    fi)
             fi));
end. {purge_text}**
```

This property is almost tautologous. It would be much more satisfying to be able to prove that the the purged version of the final labeled output is identical to the purged input. This is not possible for two reasons. Inserting the labels adds printing characters to the output which were not present in the input. Thus, given the definition of Purge_Text above, this property is not true unless one ignores the labels in the output. But there is no convenient way to distinguish labels inserted by the labeling process from identical character strings which might have appeared in the input stream.

Also, the way in which CRs and LFs are handled by the paginator potentially causes some printing characters from the input to be lost in the output. A single CR resets the current line to null, which is the paginator analog of moving the print head to the beginning of the line. However, this causes any characters on the current line to be lost. Thus, a proper new line sequence should be in the form of a LF followed by a CR. The LF causes the current line to be appended to the current page; the CR sets the current line to null, and (conceptually) positions the write head at the beginning of the line. This rather curious handling of CR is necessary to guarantee that the printing characters of the page sequence are a subsequence of the input, something that would not be true if the initial characters on a line could be overwritten following a CR.

Type Enforcement and the Labeler Environment

Proving the correct operation of the labeler is not sufficient to ensure that labeling is carried out in accordance with the TCSEC requirements. It remains to show that the labeled text is not altered before it can be output. The SAT mechanism that guarantees the integrity of such text is called the *type enforcement* mechanism and is fully described elsewhere[8, 13], so we merely summarize it here.

Associated with each object in the SAT system is a security level, an access control list (ACL), and a type. Each subject has an associated level, user, and domain. The level attributes of subjects and objects are used in enforcing the mandatory security constraints, and the user and ACL fields in enforcing discretionary access controls. The mandatory and discretionary constraints are straightforward interpretations of those mandated by the TCSEC. It is the use of the subject domain and object type fields that allows us to guarantee the integrity of the labeled text. A domain is an abstraction of the *role* that a subject is currently filling, and a type is an abstraction of the format of an object. When the labeler is executing on behalf of a particular subject that subject must be in a different domain than when executing typical user code. *Labeled text* and *unlabeled text* are of different object types. The labeler domain is afforded *read* access to unlabeled text and *write* access to labeled text, and is the only domain with write access to labeled text objects. The printer device driver is in another domain, the only domain afforded read access to objects of labeled text type; the printer domain cannot read objects of any type except labeled text. The relevant type enforcement constraints are pictured in **Figure 3-1**.

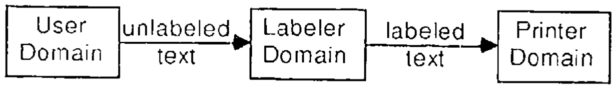


Figure 1: Information flow through the labeler domain.

Type enforcement constraints are recorded in a matrix, the Domain Definition Table (DDT), indexed on rows by domains and on columns by types. An entry in the matrix indicates whether read/write/execute access is granted a subject executing in the given domain to objects of the given type. This mechanism allows the construction of an assured pipeline[8] that maintains the integrity of labeled data. Every access is mediated by the reference monitor, which determines access rights by consulting the DDT in addition to the mechanisms for determining the mandatory and discretionary constraints.

With a DDT configured as indicated above, data of unlabeled type can be manipulated by subjects executing in user domain, but such subjects have no access to labeled data. The labeler can read unlabeled data, but write only labeled data. The printer domain permits only reading of labeled data. These constraints suffice to enforce the rule that no user process can remove or alter the labels that the labeler has inserted or signal information covertly by modifying the labeled text. Attempts to do so are violations of the type enforcement constraints encoded in the DDT and are prevented by the reference monitor. Similarly, the labeler cannot alter user files in any way. No text can bypass the labeler since the labeler domain is the only domain that can output data of labeled type and the printer domain will input only labeled text.

The type enforcement mechanism thus provides a solution to the problem of maintaining the integrity of labeled data. The solution is not at all restricted to this particular problem but rather provides the solution to a variety of similar concerns. An encryption device, for example, must be unavoidably encountered by certain types of data being propagated onto an unsecure network. This can be guaranteed using the type enforcement mechanism in an exactly analogous fashion.

The proof of the SAT type enforcement mechanism is similar to the proof of the SAT mandatory constraints and is fully elsewhere described[14, 15]. Briefly, it involves proving that the reference monitor is unavoidably consulted whenever an access is granted and that the access decisions of the reference monitor always accord with the constraints recorded in the DDT. A recent paper describes the formalization and proof of type enforcement and similar security policies in a general context[13].

CONCLUSIONS

The prototype labeler obviously does not provide all the functionality one would like in a general purpose line printer labeler. For example, the design could securely permit some additional characters to be handled, make use of the features of "smart" output devices such as resettable device parameters, and allow overstriking. Also, a more general labeler could be written with device type parameters. Such a labeler would be passed a device type and consult a table to obtain the corresponding device parameters. Labeling then would be only one part of a larger text-formatting effort, with variable results depending upon the intended target output device. This is the approach, for example, of the Scribe text formatting system[16]. The desire for such increased functionality must be weighed, however, against the additional effort that would be required for formal verification of the labeler properties.

Previous verified secure systems have been formally verified at the design level. It has been our intention to push verification of the SAT system as close as possible to the implementation level. Note that this actually provides a level of assurance beyond that required for A1 certification. The traditional view has been that code-level proofs are beyond the current state of the art in program verification. We intend to test that assertion. The labeler code, for instance, is written in executable Gypsy code and requires only a hand or mechanical

** "<:" and ":>" are the Gypsy operators which add an element onto the end of a sequence. "@" denotes sequence concatenation.

translation to the actual implementation language, a straightforward process for the constructs involved. The Gypsy Verification Environment contains mechanical tools for translating Gypsy programs to Ada or to Bliss.

The requirement that the labeler be formally verified placed constraints on its size and complexity. The proof logs of just the paginator routine and accompanying lemmas, for instance, are some 150 pages in length and the proof is rather tedious. This is more a reflection on the state of program verification than on the inherent complexity of the code. Still, increasing the functionality increases the difficulty of verification substantially. The experience gained in proving the simple prototype labeler leads us to believe that our subsequent efforts can be more ambitious. However, we are not discounting the size of the effort involved.

A labeler might take advantage of the special functionality of the intended output device. However, "smart" devices are likely to afford increased opportunities for covert channel exploitation. An output device may have internal parameters resettable via some input sequence of control characters, for example. Resetting page size may vitiate labeling constraints by placing labels outside of physical page boundaries. To avoid this interference from internal device parameters, certain *sequences* of characters would be disallowed as output from the labeler; it is much easier to limit the set of acceptable characters than to eliminate specific undesirable sequences of characters. Our approach has been to limit (by programming fiat) the range of device functionality exploitable by the user by removing all control characters. An alternative, and more likely, approach would be to insist that only devices of limited functionality be used in a secure environment thus eliminating the possibilities for abuse.

The prototype labeler is trusted only insofar as its correct functioning is crucial to the maintenance of system security, not in any special privilege it may exercise to violate constraints against information flow. Use of type enforcement limits the amount of software which must be trusted in that way, and permits the verification effort to concentrate on the functionality of trusted modules. The proofs of the integrity of the data flows between modules are trivial since they follow from the generic proof of the type enforcement mechanism.

The use of the type enforcement mechanism has proved a powerful approach to maintaining the integrity of labeled text. It allows us to provide an assured pipeline for moving unlabeled user text through the labeler to the line printer without the danger that the labels could be altered at any intermediate point. Having the type enforcement mechanism as an integral part of the security apparatus permits us to construct such an assured pipeline in any similar circumstances rather than to construct an *ad hoc* solution for each new circumstance.

Acknowledgements: This effort has been supported by U.S. Government Contracts MDA904-82-C-0144 and MDA904-84-C-6011. We wish to acknowledge the help of our colleagues at the Honeywell Secure Computing Technology Center, particularly Tom Haigh and Chuck Ferguson, and of the Gypsy group at the University of Texas, especially Don Good, Mike Smith, Bill Bevier, and Larry Akers. Our work has benefited greatly from conversations with the staff of the National Computer Security Center, MITRE, and GTE. Pat Engelking of Honeywell Communications Resources provided valuable editorial assistance.

APPENDIX: GYPSY CODE FOR THE LABELER

```
scope labeler_LP =
begin

   type LEVEL_TYPE = pending.

   type TEXT = sequence of character.

   type LINE = sequence (logical_page_width) of  character.

   type PAGE = sequence (logical page_length) of line.

   type PAGESEQ = sequence of page.

   const LOGICAL_PAGE_LENGTH    integer = pending.

   const LOGICAL_PAGE_WIDTH     integer = pending.
```

```
lemma PAGE_PARAMETERS_POSITIVE =
      logical_page_length ge 1
    & logical_page_width ge 1;

function LF: character =
begin
   exit (result = scale (10, character));
   result := scale (10, character);
end; {LF}

function FF. character =
begin
   exit (result = scale (12, character));
   result := scale (12, character);
end; {FF}

function CR. character =
begin
   exit (result = scale (13, character));
   result := scale (13, character).
end; {CR}

function SP: character =
begin
   exit (result = scale (32, character));
   result := scale (32, character);
end; {SP}

function PRINTING_CHARACTER (c: character): boolean =
begin
   exit (result iff c in (SP..'~"));
   result := (c in (SP..'~")):
end; {printing_character}

lemma SP_IN_PRINTABLE_SET =
      sp in [sp..'~];

lemma CARRIAGE_CONTROL_NONPRINTING =
      not printing_character (CR)
    & not printing_character (LF)
    & not printing_character (FF);


function N_LINE_FEEDS (n: integer): text =
begin
   entry n ge 0;
   exit (result =
           (if n = 0
              then null (text)
              else  N_line_feeds (n-1) <. LF
             fi)).
   var i: integer := n;
   result := null(text).
   loop
      if i = 0 then leave end;
      result := result <: LF;
      i := i - 1.
   end. {loop}
end; {n_line_feeds}

function N_BLANKS (n: integer). text =
begin
   entry n ge 0.
   exit (result =
           (if n = 0
              then null (text)
              else N_blanks (n-1) <: SP
             fi));
   var i: integer := n;
   result := null(text);
   loop
      if i = 0 then leave end;
      result := result    SP;
      i := i - 1.
   end; {loop}
end; {n_blanks}

function N_NULL_LINES (n: integer): page =
begin
   entry n ge 0.
   exit (result =
           (if n = 0
              then null (page)
              else N_null_lines (n-1) <: null(line)
             fi));
   var i. integer := n.
   result := null(page).
   loop
      if i = 0 then leave end.
      result := result <  null(line).
      i := i - 1.
   end. {loop}
end. {n_null_lines}
```

```
procedure PAGINATOR (inseq: text;
                     var purgetxt: text;
                     var pages: pageseq) =
begin
   exit   purgetxt = purge_text (inseq)
        & correct_page_sequence (pages);
   var current_column_position: integer := 1;
   var current_row_position: integer := 1;
   var current_input_position: integer := 1;
   var current_page: page := null(page);
   var current_line: line  = null(line);
   pages := null (pageseq);
   purgetxt := null (text);
   loop
     assert
          purgetxt
        = purge_text (inseq[1 .. current_input_position - 1])
      & correct_page_sequence (pages)
      & correct_partial_page (current_page, current_line,
                              current_row_position,
                              current_column_position)
     & size (current_page) = current_row_position - 1
     & size (current_line) = current_column_position - 1
     & current_row_position in [1..logical_page_length]
     & current_column_position in [1..logical_page_width];
      if current_input_position = size(inseq) + 1
         then leave
      end;
      if inseq (current_input_position) = CR then
         current_line := null(line);
         current_column_position := 1;
      else
         if inseq (current_input_position) = FF then
            current_page := current_page <: current_line
                 @ N_null_lines (logical_page_length
                                 - current_row_position);
            pages := pages <: current_page;
            current_page := null(page);
            current_row_position := 1;
            current_line := null(line),
            current_column_position := 1;
         else
            if (inseq (current_input_position) = LF) then
               current_page := current_page <: current_line;
               if current_row_position = logical_page_length
               then
                  pages := pages <: current_page;
                  current_page := null(page);
                  current_row_position := 1;
               else
                  current_row_position
                          := current_row_position + 1;
               end;
               current_line := null(line)
                   @ N_blanks (current_column_position - 1);
            else
               if printing_character
                    (inseq (current_input_position)) then
                  purgetxt := purgetxt
                          <: inseq (current_input_position);
                  current_line := current_line
                          <: inseq (current_input_position);
                  if   current_column_position
                     = logical_page_width
                  then
                     current_page := current_page
                                     <: current_line;
                     current_line := null(line);
                     current_column_position := 1;
                     if   current row position
                        = logical_page_length
                     then
                        pages := pages <: current_page;
                        current_page := null(page);
                        current_row_position := 1;
                     else
                        current_row_position
                                := current_row_position + 1;
                     end;
                  else
                     current_column_position
                             := current_column_position + 1,
                  end; {if}
               end; {if}
            end; {if}
         end; {if}
      end; {if}
      current_input_position := current_input_position + 1;
   end; {loop}
   current_page := current_page <: current_line
        @ N_null_lines (logical_page_length
                        - current_row_position);
   pages := pages <: current_page;
end; {paginator}

function PURGE_TEXT (inseq: text): text =
begin
   exit (assume result =
          (if inseq = null(text)
              then null(text)
              else
                 (if printing_character (last(inseq))
                     then  purge_text (nonlast (inseq))
                                          <: last(inseq)
                     else purge_text (nonlast (inseq))
                  fi)
           fi)).
end; {purge_text}


function CORRECT_LINE (ln: line): boolean =
begin
   exit (assume result iff
          (  size(ln) le logical_page_width
           & (all i: integer,
                 i in [1 .. size(ln)]
              -> printing_character (ln[i])))).
end; {correct_line}

function CORRECT_PARTIAL_LINE (ln: line;
                               current_col_position: integer)
             : boolean =
begin
   exit (assume result iff
          (  current_col_position in [1..logical_page_width]
           & correct_line (ln)));
end; {correct_partial_line}

function CORRECT_PAGE (pg: page): boolean =
begin
   exit (assume result iff
          (  size(pg) = logical_page_length
           & (all i: integer,
                 i in [1 .. size(pg)]
              -> correct_line (pg[i])));
end; {correct_page}

function CORRECT_PARTIAL_PAGE (current_page: page;
                               current_line: line;
                               current_row_position: integer;
                               current_col_position: integer)
          : boolean =
begin
   exit (assume result iff
         (  correct_partial_line (current_line,
                                  current_col_position)
          & current_row_position in [1..logical_page_length]
          &   current_row_position
            = size( current_page @ [seq: current_line])
          & (all i: integer,
                i in [1 .. size(current_page)]
             -> correct_line (current_page[i])));
end; {correct_partial_page}


function CORRECT_PAGE_SEQUENCE (pages: pageseq): boolean =
begin
   exit (assume result iff
          (  pages = null(pageseq)
           or
             (  correct_page (last (pages))
              & correct_page_sequence (nonlast (pages))));
end; {correct page sequence}


lemma PURGEABLE_CHARACTER_EXTENSION_LEMMA (inseq: text;
                                          c: character) =
      not printing_character (c)
  ->  purge_text (inseq) = purge_text (inseq <: c).


lemma NONPURGEABLE_CHARACTER_EXTENSION_LEMMA (inseq: text;
                                             c: character) =
      printing_character (c)
  ->  (purge_text (inseq) <: c) = purge_text (inseq <: c);

lemma SIZE_N_NULL_LINES (n: integer) =
   n ge 0
 ->
   size(n_null_lines (n)) = n;

lemma SIZE_N_BLANKS (n : integer) =
    n ge 0
  ->
    size (n_blanks (n)) = n;
```

```
lemma N BLANKS_ALL_BLANK (n. i  integer) =
     (n ge 0 & i in [1  n])
  -> n_blanks(n)[i] = sp.

lemma N_NULL_LINES_ALL_NULL (n. i  integer) =
     ( i in [1  n] )
     ->
     n_null_lines(n)[i] = null(line).

lemma LINE_INDEX_LEMMA1 (ln. more   line. i   integer) =
   i in [1  size (ln)] -> (ln @ more)[i] = ln[i].

lemma LINE_INDEX_LEMMA2 (ln. more  line; i . integer) =
   i in [size(ln)+1  size(ln)+size(more)]
     ->
   (ln @ more)[i] = more [i-size(ln)].

lemma TEXT_INDEX_LEMMA1 (txt.more   text; i integer) =
   i in [1..size(txt)] -> (txt @ more)[i] = txt[i].

lemma TEXT_INDEX_LEMMA2 (txt,more  text. i .integer) =
   i in [size(txt)+1  size(txt)+size(more)]
     ->
   (txt @ more) [i] = more [i - size(txt) ]

lemma SEQUENCE_INDEX_LEMMA1 (ppg. more: page.
                             i  integer) =
    i in [1. size(ppg)]
  ->
    (ppg @ more)[i] = ppg[i];

lemma SEQUENCE_INDEX_LEMMA2 (ppg. more   page;
                             i  integer) =
    i in [size(ppg)+1..size(ppg)+size(more)]
  ->
    (ppg @ more)[i] = more[i-size(ppg)].

lemma SEQUENCE_ELEMENT_LEMMA (elem  line; ppg .page) =
    elem in ppg
     iff
    some i :integer.  i in [1. size(ppg)]  &
                      elem = ppg[i];

lemma EXTEND_TO_PAGE (current_page   page.
                      current_line   line.
                      current_row_position.
                         current_column_position . integer) =
    correct_partial_page (current_page. current_line.
                           current_row_position.
                           current_column_position)
  -> correct_page (  current_page <  current_line
                   & n_null_lines (  logical_page_length
                                     - current_row_position)).

lemma ADD_CORRECT_PAGE (pages   pageseq.
                        current_page . page.
                        current_line   line.
                        current_row_position.
                           current_column_position   integer) =
      correct_page_sequence (pages)
    & correct_partial_page (current_page. current_line.
                             current_row_position.
                             current_column_position)
  -> correct_page_sequence
      (  pages @ [seq  current_page @ [seq  current_line]
           @ n_null_lines (logical_page_length
                             - current_row_position)]);

lemma EXTEND_LAST_LINE (current_line  line.
                        current_page. page.
                        pages  pageseq.
                        c  character) =
      correct_page_sequence
          (pages <  ( current_page <. (current_line)))
    & printing_character (c)
    & size (current_line) + 1 le logical_page_width
   -> correct_page_sequence
          (pages <  (current_page <  (current_line <  c))).

 procedure LABELER (inseq  text.
                    var purgetxt  text.
                    var outseq  text.
                    lvl  level_type)
 begin
    exit  purgetxt = purge_text (inseq)
        & correctly_labeled (outseq. lvl).
    var pages. pageseq
    var label  text.
    label  = associated_label (lvl).
    paginator (inseq. purgetxt. pages).
    outseq   null(text).
```

```
   loop
      assert  correctly_labeled (outseq. lvl)
            & label = associated_label (lvl)
            & purgetxt = purge_text (inseq)
            & correct_page_sequence (pages).
      if pages = null(pageseq) then leave, end;
      outseq .= labeled_page (last (pages). label) @ outseq.
      pages .= nonlast (pages).
   end, {loop}
end: {labeler}

function ASSOCIATED_LABEL (lvl: level_type). text =
begin
   exit (assume size (result) le logical_page_width).
   pending;
end; {associated_label}

function LABELED_PAGE (pg  page; label: text): text =
begin
   entry correct_page (pg).
   exit correctly_labeled_page (result. label);
   var i. integer  = 0.
   result .= [seq. FF. LF. CR] @ label @ [seq: LF, CR];
   loop
      assert  correctly_labeled_partial_page
                             (result. label. i)
            & correct_page (pg)
            & i in [0..logical_page_length].
      if (i = logical_page_length) then leave, end;
      result .= result @ pg(i+1) @ [seq. LF. CR].
      i := i + 1;
   end; {loop}
   result := result @ [seq  LF. CR]
                    @ label @ [seq: LF. CR].
end; {labeled_page}

function CORRECTLY_LABELED (outseq  text.
                            lvl  level_type)  boolean =
begin
   exit (assume result iff
           (if (outseq = null(text))
               then true
               else
                  (some pg. text, some outseq2. text,
                    (  (outseq = pg @ outseq2 )
                     & correctly_labeled_page
                         (pg. associated_label (lvl))))
            fi)).
end; {correctly_labeled}

function CORRECTLY_LABELED_PAGE (pg. text.
                                 label. text)  boolean =
begin
   exit (assume result iff
           (some body  text.
             (   (pg = [seq  FF. LF. CR] @ label
                         @ [seq  LF. CR]
                         @ body @ [seq  LF. CR] @ label
                         @ [seq  LF. CR])
              & correct_page_body
                   (body. logical_page_length)))).
end, {correctly_labeled_page}

function CORRECTLY_LABELED_PARTIAL PAGE (outseq  text.
                                         label  text.
                                         i: integer)
             . boolean =
begin
   exit (assume result iff
          (some body. text.
            (  (outseq = [seq. FF, LF, CR] @ label
                           @ [seq  LF.  CR] @ body)
             & correct_page_body (body. i)))).
end. {correctly_labeled_page}

function CORRECT_PAGE_BODY (body  text. pagesize  integer)
             boolean =
begin
   exit (assume result iff
          (if pagesize = 0
              then true
              else
                 (some ln  text. some body2  text.
                   (   (body   body2 @ ln @ [seq  LF. CR])
                     & correct page body
                          (body2. pagesize   1)
                     & correct line (ln)))
           fi)).
end. {correct page body}

end. {scope}
```

References

1. Department of Defense, "*Trusted Computer Systems Evaluation Criteria*", CSC-STD-001-83, August 15, 1983.

2. Rudell, Mindy, "Labeling Screen Output", *Proceedings of the Symposium on Security and Privacy*, IEEE, 1985, pp. 237-240.

3. Kurth, Helmut, "Paper Output Labeling in a Dedicated System Running under MVS", *Proceedings of the 8th National Computer Security Conference*, NBS, 1985, pp. 86-90.

4. McHugh, John, "An Emacs-Based Downgrader for the SAT", *Proceedings of the 8th National Computer Security Conference*, NBS, 1985, pp. 133-136.

5. Vickers Benzel, T., and D.A. Tavilla, "Trusted Software Verification", *Proceedings of the Symposium on Security and Privacy*, IEEE, 1985, pp. 14-31.

6. Lampson, Butler, "A Note on the Confinement Problem", *Comm. of the ACM*, Vol. 16, No. 10, October 1973, pp. 613-615.

7. Bell, D.E., and L.J LaPadula, ""Secure Computer System. Unified Exposition and Multics Interpretation"", Tech report MTR-2997, MITRE Corp., July 1975.

8. Boebert, W.E., R.Y. Kain, "A Practical Alternative to the Hierarchical Integrity Policies", *Proceedings of the 8th National Computer Security Conference*, NBS, 1985, pp. 18-27.

9. Kain, Richard Y., W. Earl Boebert, "Domains and Role Enforcement", to appear..

10. Good, Donald I., 'SCOMP Trusted Processes", ICSCA Internal Note 138, The University of Texas at Austin.

11. Good, D.I., R.M Cohen, C.G. Hoch, L.W. Hunter, and D.F. Hare, "Report on the Language Gypsy, Version 2.0", Tech. report ICSCA-CMP-10, Institute for Computing Science, University of Texas at Austin, September 1978.

12. Good, D.I., B.L. Divito, M.K. Smith, "Using The Gypsy Methodology", Tech. report, Institute for Computing Science, University of Texas at Austin, June 1981.

13. Haigh, J.T., W.D. Young, "Extending the Non-Interference Version of MLS for SAT", *Proceedings of the 1986 Symposium on Security and Privacy*, IEEE, 1986, pp. 232-239.

14. Boebert, W.E., W.D. Young, R.Y. Kain, S.A. Hansohn, "Secure ADA Target: Issues, System Design, and Verification", *Proc. Symposium on Security and Privacy*, IEEE, 1985

15. Young, W.D., W.E. Boebert, R.Y Kain, "Proving a Computer System Secure", *Scientific Honeyweller*, Vol. 6, No. 2, July 1985, pp. 18-27.

16. Reid, Brian K., *Scribe: A Document Specification Language and its Compiler*, PhD dissertation, Carnegie-Mellon University, October 1980.

LIMITATIONS OF DIAL-UP SECURITY DEVICES

Eugene F. Troy, CDP
National Bureau of Standards
Institute for Computer Sciences and Technology
Building 225, Room B-266
Gaithersburg, MD 20899
(301) 921-3551

ABSTRACT

A number of hardware devices intended to improve dial-up communications security have recently been introduced to the commercial market. These devices can be separated for discussion into six major groups, according to their primary protection objective. The six groups are: host port protection devices, user terminal security modems, user authentication devices, terminal identification devices, line encryption devices, and message authentication devices.

Many claims have been made about the degree of protection afforded by these mechanisms. In contrast, there are persistent rumors from the "hacker underground" that the security of some of these devices can be broken. Also, several problems have been identified in administering this family of devices, some of them economic or practical and others directly related to security. This paper reviews the classes of devices available, describes their basic characteristics via examples, discusses typical security flaws and implementation weaknesses, and recommends a series of approaches to overcome these problems.

STATEMENT OF THE PROBLEM

Almost every computer of any size has one or more ports which are connected via modems to the public telephone system (POTS). It has become a popular hobby of teenagers and others to identify these computers and explore them in various ways, some of which are disruptive to business operations. In recent years, there have been growing indications that less savory individuals, such as spies and criminals, are using the same techniques as "hackers", penetrating computer systems in order to steal valuable information or to defraud organizations. This paper makes no distinction of motives, referring to all instances of attempted or actual access by unauthorized persons as "penetrations" and the persons themselves as "intruders".

To counter this threat, good access control security is now mandatory for any computer system connected to POTS. In most cases, the computer's operating system can provide an adequate level of access control if its security-related features are used properly. However, many smaller operating systems do not provide these features, and many more systems are improperly administered to the extent that severe weaknesses exist. If this security is not available through use of the computer system's own capabilities, then specialized hardware security devices may be used to augment or supplant the security features and provide the necessary level of protection.

These hardware devices used for dial-up security are a mixed blessing. Technical weaknesses in the design and implementation of some of these products may exist which in themselves offer the intruder an avenue of approach, effectively negating their usefulness. In addition, there often are administrative drawbacks to the use of these security devices, in the form of unjustifiable extra costs and administrative burdens. Potential and current users of the dial-up security devices need to examine these weaknesses and drawbacks carefully so that security and effectiveness may be improved by correct usage of the devices.

NATURE OF THE THREAT

There is no doubt of the growing penetration threat to computers with dial-up access to the POTS. A number of factors increase that threat to the point where it must be taken seriously but without over-emphasis.

Openness of Dial-Up (POTS) Network. For several years, it has been possible for anyone with access to a telephone connected to POTS to dial directly almost anyone else with the same access in this country and most other countries of the free world. The only impediment to this access is knowledge of the target's telephone number. A very small number of protective measures are available in some locations for POTS, but these are costly and not well known. These measures are: unlisted numbers, automatic call-tracing, and limited call-in list. In general, anyone with access to a telephone and a modem-equipped terminal anywhere in the world has the potential to become a user of any computer with dial-up access in the world. It is known that some of the most sophisticated penetration attempts on computers in the United States have come from Europe and the Middle East.

Availability of Penetration Equipment. Anyone with a minimal grasp of present computer technology can readily understand that no complicated equipment is needed for dial-up penetration. Terminal emulation software is readily available for all personal computers, including those in the inexpensive hobby class. Likewise, modems can be obtained at any computer or electronics store, with starting prices at less than $100. One of the most commonly used penetration instruments is the Commodore 64, a hobby computer for which extremely sophisticated "hacker" software has been written and is available on pirate bulletin boards.

Intruder Understanding of Technology. Modern hobbyists have a grasp of computer and communications technology that is little short of awesome in some cases. It must be accepted as a guiding rule that intruders, be they "hackers" or more serious criminals, know at least as much about technology as anyone within the target organization. The only facts that they might not know are the specific details of system implementation in a particular organization. It appears that disgruntled employees or other insiders have provided even this information to pirate bulletin boards and other underground sources.

Sharing of Penetration Information. The pirate bulletin boards are an indication that intruders like to share information and brag about their exploits. Often, this is the primary avenue for others to obtain a good education about the technology and security protection methods commonly used. This widespread sharing of information significantly increases the level of intruder threat.

NATURE OF VULNERABILITIES

There are some common vulnerabilities in the operation or administration of computer systems which make it much easier for intruders to gain telephone access. It is a sad commentary that by far the greatest majority of known penetrations have occurred by simple exploitation of prevalent administrative weaknesses, and not from any technical sophistication on the part of the intruder.

USERID/Password Administration. The typical penetration attempt starts out by using USERIDs and passwords that intruders know to be commonly used in poorly secured systems. Pirate bulletin boards often provide lists of them for novices. The most notorious examples include the following.

- **Any vendor-supplied USERID or password** (the most common and effective penetration avenue of all, because these typically carry "super-user" privileges).
- **Common first or last names** (penetrators often obtain organization telephone books and try likely names).
- **Any common abbreviation,** especially computer-related.
- USERID = **password.**
- **One or two letters or numbers.**
- **Any word in a dictionary** (intruders are now harnessing on-line dictionaries or spelling checkers to their penetration software).

24-hour Dial-Up Accessibility. It is remarkable how many computer systems of all sizes permit dial-up access at all hours, even though it may be unlikely that any legitimate users may be seeking access outside of normal weekday business hours. This is coupled with the fact that most penetration attempts occur during non-business hours. Often, the remedy is extremely simple: turn off or disconnect modems when not actually needed.

Operating System Weaknesses. Many operating systems either do not have many security features or, more typically, provide the features but make them optional to the using organization. Often, the features are viewed by system software engineers as unnecessary or as causing reductions in system performance or ease of use. The latter may be true, but the threat from intruders is growing to such a degree that it is almost irresponsible to operate a system with dial-up access that does not have demonstrably effective access control.

BASIC DIAL-UP SECURITY REQUIREMENTS

In order to reduce the effectiveness of intruder penetration via the telephone system, there are four basic requirements which should be met. Typical mainframe and minicomputer operating systems, when properly used, may be able to take care of all or part of the problem, but no unadorned microcomputer operating system can do so. If these requirements are not adequately met by the host itself, then add-on equipment may be needed to supplement its protection.

User Identification and Authentication. This is the keystone of all access control security. A well administered USERID and password process is very important for computers with dial-up access, because it is the first access control mechanism typically encountered as the user enters a system. When this capability is weak or nonexistent for any reason, a variety of external hardware mechanisms can provide or augment this capability.

Security Event Logging. It is now an accepted security principle that all dial-up communications activity between host and user ought to be monitored in order to uncover intrusion attempts, or worse, successes. For larger computers, this can be done routinely by the system journal. Several add-on external devices can perform this function as part of a dial-up user access control strategy.

Limiting the Attacks. If the intruder does not know the correct access codes, then he must make many guessing attempts. In some cases, this is done by the intruder's computer, which runs a program that generates and tries a series of passwords one after another. Any mechanisms that limit the number or speed of repetitive user sign-on attempts per dial-up connection can help counter this type of attack.

Concealment of Information. If the information which is accessible via dial-up connection is very confidential or susceptible to fraud, then it may need to be protected from disclosure or tampering via wire taps or other forms of interception. Any mechanisms or software that encrypt the information on the line can help prevent this condition.

DIAL-UP SECURITY DEVICES AVAILAI

To protect dial-up communications with hardware security devices, the communications link itself is secured independently, external to the computer hardware or software. Several types of devices are available that apply one or more of the dial-up protection functions described above to the communications link.

BENEFITS OF SECURING DIAL-UP LINKS

The primary advantage in using hardware security devices is that it reduces the degree of dependence on other software or procedural security mechanisms in the system. Many of those mechanisms may not be strong enough or may not even be readily available for a specific computer system. There are two other notable benefits to be gained by applying hardware protection to the communications link.

Separation of Function. In using hardware security devices, separation of function is gained by:

- **Externalization** of a set of security functions outside the machine, physically and logically separated from the host.

- **Kernelization** of a portion of the security function into a single dedicated mechanism for reduced and controlled access via communications.

Additional Layers of Protection. Hardware security devices on the system's communications links provide formal protection of the network itself. Most hardware protection is designed to control authorization to a single system object, the communications port. Other software and procedural security mechanisms should still be used to reduce logical exposure to the remainder of the system.

THE SIX TYPES OF HARDWARE

In protecting any set of dial-up communications ports, two basic approaches can be taken which involve adding hardware protective devices to the dial-up circuit. These approaches are referred to as the "one-end" and "two-end" solutions, depending upon the placement and configuration of the protective hardware.

The "One-end Solution" -- Two Types. This solution provides a separate password on the communications link itself, by using hardware to protect only one end of the communications link. Two types of devices are available, one for installation on the host computer and the other on the user's terminal. These devices perform a basic user authentication screening function, normally without the requirement for users to obtain any extra equipment.

The "Two-end Solution" -- Four Types. More security is gained by using a matched set of hardware protective devices for both ends of the dial-up circuit (computer and terminal). These devices can communicate with each other in various ways to perform their security functions.

The four types of equipment are divided up by function. Three perform authentication functions, respectively, of the user, the user's terminal or location, and the message or data transmitted via the circuit. The fourth type is line encryption, which performs a concealment function on the transmitted data, and may also be construed as authenticating the user or originating terminal via the process of encryption key exchange.

"ONE-END" DEVICE FEATURES

The first group of devices to be discussed improves user access control by performing a preliminary call-screening or authentication function. Typically, such a device is totally independent of the computer. Devices in this category are called "one-end solutions", because they are used on only one end of the communications circuit between the host and terminal, but not both. Most versions of one-end protection devices are installed at the host computer end, but some newer multi-function devices are connected to the user's terminal.

Host Port Protector Devices (PPDs). A PPD is fitted to the communications port of a host computer, providing the function of authorizing user access to the port itself, prior to and independent of the computer's own access control functions. It is specifically designed to help control terminal access when dial-up communications are used.

Depending on design, a PPD may operate between the host and modem (digital side), or it may operate between modem and telephone set (analog side). Some modems include PPD functions in a single unit. Once connection and user validation take place, the PPD becomes passive in the circuit. The four primary features of PPDs are described below.

- **Password Tables.** All PPDs require the user to enter a separate authenticator or password in order to access the computer's dial-up ports. This set of external password tables independent of the computer's operating system is the primary protection given by PPDs. All of these devices limit the number of sign-on attempts per telephone connection, in order to deter repetitive attacks.

- **Call-back to Originator.** Most PPDs do not have or need this capability, although some persons erroneously call all PPDs "call-back devices". This feature, when present, is used as a second level of external user authentication. A PPD with call-back will ordinarily require the user to enter a PPD table password, and then will disconnect the line. The PPD then identifies the user's telephone number that matches the password and makes a return call to the user for host connection.

- **Hiding the Port.** All PPDs have some ability to "camouflage" the computer's dial-up ports so that the computer cannot be identified by an unauthorized caller. Some

PPDs located on the "analog-side" use a synthesized human voice to hide the modem tone on initial connection. "Digital side" PPDs send their own screen displays via the modem to the user's terminal which masks the kind of computer they are protecting, vital information needed by the intruder to carry out his attack.

● **Attack Signalling.** Most PPDs are able to provide some form of warning signals or records of dial-up attack. Some models use front-panel display lights, others maintain internal logs in RAM storage, and the most expensive models use the disk storage of dedicated personal computers to record many types of information about communications activity.

Security Modems for Users. Several new devices are part of the trend towards integration of security features into standard devices. Controlled-access "security modems", installed on user terminals, are single-user modems which incorporate a set of outbound call-screening security functions to control access to the host from the user end.

Security modems will not make the dial-out connection until the user enters a specified password. Inside the modem, these passwords are matched in a secured table with dial-out telephone number sequences necessary to connect the user to specified host computers. The table also can contain a complete log-on sequence for transmission to the host once connection is made, but it is advisable not to include the log-on password in this sequence.

"TWO-END" DEVICE FEATURES

In higher-security systems, password protection of the port may still seem inadequate. A more positive identification of the specific terminal or user may be desired. A measure of resistance to snooping or tampering with communications traffic may also be needed. The "two-end" approach makes use of a security device at the user terminal end which matches to a device or special software at the host computer. The four types of devices that belong to the two-end solution family are described below.

User Authentication "Tokens". Some two-end devices perform highly secure authentication of individual system users. These devices are based on the concept of a unique "token" to be used somewhat like a mechanical password. A token is a small item, such as a plastic "smart-card", given to each authorized system user that must be used to gain access to the system. Each token has a special algorithm or some other unique and non-copyable identifier embedded in it. The host computer can identify the user uniquely by means of the token's distinctive characteristics.

Most varieties of user authentication tokens are hand-held and require no terminal attachment. This type of token may take various forms. Some examples now on the market include a calculator with special circuitry, a "smart" plastic card which displays a time-based authenticator continuously, and a lightsensitive wand which is designed to read and interpret special terminal challenge displays sent by the host.

For most tokens, the user must enter into the token some challenge information sent by the host. A liquid crystal display (LCD) on the token then shows the computed result of the challenge. The user must enter this authentication information via the terminal. The host reads the authentication information and compares it to the "right" answer it has previously generated and then decides whether to approve access.

Terminal Authentication Devices. The second type of device in the two-end solution family is designed to authenticate a specific user terminal. Terminal authentication devices work very much like user authenticators. They use matching pairs of devices inserted in the communications circuit. One device is placed between the terminal and modem, and the other is attached to the host computer's port. A typical product includes a four-port unit for the host end which is able to generate challenges to the small portable units that connect to the terminals. Each terminal unit is uniquely encoded for identification by the host unit.

Hybrid versions of terminal authenticators are also available, which include the capability to authenticate each user at the same time. For example, a newer version of the terminal unit just described has a slot where each user is to insert their own token in the form of any pre-validated magnetic striped card (even a bank or charge card). Another popular product takes a similar approach, requiring each user to insert their own thick plastic card with embedded identification circuitry into the unique terminal unit. Both of these products automatically accept the challenge from the host, use the algorithm or data in the user's token to perform the required calculations, and then transmit the results to the host for verification.

Line Encryption Devices. Encryption is the process of "scrambling" information in a pre-determined way so that it is unintelligible to anyone who does not know how to "unscramble" it. Encryption is the highest form of security which can be applied to dial-up communications, because it has several attributes which cover most communications security needs.

First, the primary rationale for using encryption is that it conceals the information passing over the communications link from disclosure to snoopers. Second, encryption in some modes can assure the integrity of the message, so that tampering or transmission errors can be identified. (Note that the process of message authentication, to be discussed next, is better for assuring message integrity.) Third, the uniqueness of the encryption key which must be shared by sender and receiver enforces an extremely high degree of user authentication. If both sender and receiver share a single key, they must have exchanged it or been assigned it by a third party.

Encryption devices can take two forms. In the more traditional form, the circuitry is enclosed in a small box that is connected in series between the port and the modem, on either end of the communications circuit. In the newer form, designed for personal computers, all circuitry is contained on a single circuit board that is plugged into one of the standard slots inside the computer. With the latter form, it is usually also possible to use the circuit board for encryption of internal dis[illegible] files, [illegible] addition to using it for communi[illegible]tions. With either form, the host's comm[illegible]cations ports are fully protected from in[illegible]ders.

Newer [illegible] more sophisticated encryption devices ca[illegible] linked together so that they automati[illegible]ly identify each other and exchange [illegible]ion encryption keys in a secure way witho[illegible] need for human intervention. Thi[illegible] tal [illegible]re of the key management problem which h[illegible] roubled encryption users from antiquity

Messag[illegible] Data Authentication Devices. This a[illegible]ach, designed originally for electro[illegible] funds transfer (EFT), can readily be use[illegible] verify the integrity of any colle[illegible]n of data being transmitted or stored [illegible]d to ensure that it is not altered witho[illegible] being detected. In the usual communi[illegible]ations application, a device uses a pre-spe[illegible]ified key to encrypt selected fields in a formatted message. Alternatively, the device may be set to encrypt the complete contents [illegible] any message or data file via the key.

The device uses the encrypted text to form the "message authentication code" (MAC), a cryptographic checksum which it then appends to the clear-text message or data file to serve as a signature or seal. The recipient must have an identical device which checks the seal by duplicating the original MAC generation process with the same key. Communications links protected full-time by message authentication devices would be highly resistant to intruders.

TYPICAL DEVICE WEAKNESSES/DRAWBACKS

The good news is that the devices described in the previous section can significantly improve a system's resistance to dial-up penetration. The bad news is that there is no free lunch, as the saying goes. There is always a set of negative aspects to be considered in selecting new products. The devices may have weaknesses which could in turn be exploited by intruders, and there are always some administrative drawbacks in terms of costs and inherent usage problems.

This section discusses these negative issues in order to help the system security manager decide whether to use the devices at all and evaluate which of them might be most beneficial. It is important to note that the weaknesses or drawbacks discussed here are not applicable to all devices or models.

TECHNICAL WEAKNESSES

The technical weaknesses of dial-up security devices include vulnerabilities in the way specific devices are designed or the way they are used. It is clear that adding security devices to a system will increase the security only if the mechanisms are not themselves flawed and if they are used properly. A practical analysis of dial-up security devices indicates that in the worst case they could even degrade security by inducing a greater degree of trust than warranted.

The following discussion is presented so that the potential or current user of these devices can better evaluate the device characteristics required in a particular application, be better prepared to ask penetrating questions of device vendors, and be more cautious about relying upon the devices too heavily. This information could also be used for background purposes in framing equipment selection criteria.

Design Weaknesses. These weaknesses are security flaws inherent in the design of the protection device. There are known instances in which intruders have defeated certain types of the devices because of the way they operate or are used.

- **Extra Passwords or Tokens.** Most of the devices which perform a direct user authentication function have the inherent weakness of requiring the user to remember or carry an additional authenticator (password or token) beyond those already needed for system access. Many users have trouble remembering their ordinary passwords, and will commonly resort to writing them on the terminal or keeping them nearby. Additional required passwords will tend to amplify this problem by making the exposure to surreptitious password discovery greater than it already is. There is a tendency to treat tokens in a similarly insecure manner by leaving them near the terminal where they will be handy, instead of carrying them on the person and risking the possibility that they will be forgotten at home.

- **Weak Password Mechanisms.** Adding port-level passwords to a system with weak logical access control procedures does not in itself assure significantly better security. The improvement in security must come from effective password management procedures, wherever they are used. Hopefully, these procedures can be enforced or at least supported by the device using them. The design of devices presently on the market makes it very easy to assign port-level passwords with weak structures to users and then not change them when needed. None of the devices has a way of identifying weak or even repetitive passwords. None provides for dating of the passwords to determine their age or otherwise provide for mandatory change. None forces the security administrator to purge the vendor-supplied master password from the system. Only the call-back feature is any protection against users sharing the same password.

• **Security Event Logging.** It was pointed out earlier that good security event logging is important to assure effective dial-up security by countering penetration attacks. Many of the add-on hardware security devices do not provide this capability at all or do so in rudimentary fashion. Others do not store the information collected in readily available or easy to use form. This must be considered a weakness.

• **Call-back Interception.** The call-back feature on PPDs which use it can be a mixed blessing. Modern dial-up intruders typically have access to many of the techniques used by "phone phreaks" to trick telephone line control devices. Some of these tricks have even been programmed into the hacker software packages now available via pirate bulletin boards and other sources. Call-back relies on the ability of the PPD to drop the incoming line and initiate a new call to the potential user. If the intruder can trick the PPD into falsely sensing a line disconnect, then he can stay on the line and thwart the intent of the PPD's call-back attempt. This particular trick will only work when the PPD's incoming and outgoing lines are the same for a particular call and the intruder has already identified a valid password.

Some PPDs have one set of lines for screening incoming calls and another set of lines used only for making the call-back connection. This design approach could make the intruder's penetration even easier if the PPD is not able to recognize an incoming ringing signal on its outgoing lines. The intruder has only to guess one of the outgoing telephone numbers, given a particular incoming number, then call the outgoing line and "camp" on it with a ringing signal. When the PPD attempts to call out and make connection with a user, the intruder is there waiting to intercept the call. Vendors who make use of call-back should be closely questioned to determine whether their devices can defend against attacks such as those discussed.

• **Password Table Security.** Devices which use passwords have a variety of procedures and security features for administering the password tables. If this security can be penetrated by an intruder, then the device has been nullified. Some devices permit any terminal connected to their incoming port to gain access to the tables, usually by furnishing a form of supervisory password. Other devices with greater security may require the supervisor's terminal to use a special port. Some may permit entry into table-changing mode only when a standard brass key is inserted into a master switch. Table-changing procedures should be evaluated carefully in terms of the degree of security improvement desired.

• **Penetration and Bypass.** If an intruder (including an insider) can gain physical access to the security device, and even worse, if he can open it up, then it may be an easy matter to nullify or bypass it. Only one port protection device now on the market stresses physical impenetrability and has a disconnect alarm. Others have varying degrees of physical hardness, usually low. All communications security devices should be protected by restricted physical access.

• **Camouflaging (sign-on clues).** Dial-up security devices are still not very well known to the hacker community, mainly because not very many of them are being used yet. Few attack techniques against specific models appear to have been developed up to this time. Such will almost certainly __not__ be the case in the future. Computer hobbyists of all ages have repeatedly demonstrated that professional software and hardware designers have no monopoly on insight, ingenuity and innovation.

Some of the PPDs do not permit the security administrator to change the sign-on screens or other initial presentation features. Being able to change them would help obscure the identity of the device itself from an intruder.

__Implementation Weaknesses.__ Weaknesses of implementation are probably more important in a practical sense than weaknesses of design. If a security device is not used properly, it may constitute a greater risk than if it is not used at all. Following are a number of typical implementation weaknesses associated with some dial-up security devices.

• **Password/Authenticator Problems.** One-end devices make use of passwords to identify valid users. These passwords are subject to the same types of administrative problems as passwords used with the operating system or applications. All password-oriented devices presently available require the administrator to assign and manage the passwords, which may result in the following example problems: Passwords may be trivial in length or construction, so they can be easily guessed; passwords may not be removed when no longer needed; passwords may not be changed frequently enough; the device vendor password may be retained, with its attendant supervisory level of privileges; passwords may be exchanged between users or shared with those who have none assigned. This reduces individual accountability to near zero. A similar set of problems applies to the use of encryption without good key management, as well as to user authentication tokens.

• **Incompatibilities With User Terminals.** Some PPDs require a telephone handset touch-pad or the human voice to enter the user authentication information. Other protection approaches, such as external encryptors and terminal authenticators, require the user to place a security device in series between the terminal port and the modem. Unfortunately, it is rapidly becoming commonplace for user terminals (including portable and personal computers) to be directly connected to the telephone system without voice handsets or external modems or both. If this is the case, then the security devices will not work without re-engineering the connection between the terminal and the telephone system. This will typically require some amount of user equipment replacement.

• **User Needs vs. Enforcement Ability.** Installing security hardware in the dial-up circuit tends to induce rigidities in the ways that users are able to interface with the host system. In order for some types of users to connect properly, some of the desired security features may have to be overridden. For example, PPDs with the call-back feature enabled require the users to call from a fixed set of terminal telephone numbers. This is impractical if the users, such as traveling salespeople, are on the road or use more than one telephone number for their terminals. The call-back feature can usually be selectively disabled for these people, but then some of the user codes are in effect less secure than others.

This same situation could hold true if the user security device were in the form of a box which must be inserted in series between the terminal and the modem, such as a terminal or message authenticator or an encryptor. This configuration might not match the types of terminals and modems some users have, and would require special (often less secure) procedures for them.

ADMINISTRATIVE DRAWBACKS

In addition to the technical problems which may exist in using dial-up security devices, there are a number of serious administrative concerns that should be examined before this equipment is obtained. These concerns boil down to money and the problem of living with the devices once they have been placed into operation.

Cost Factors. The basic money issue with respect to dial-up security devices is the question whether they are cost-effective in reducing penetration risks. There are a number of cost factors involved, not all of them obvious or easy to calculate. When these are added together for a particular application, the cost for communications protection via hardware may become very high. Here are some of the most significant cost factors to consider.

- **Hardware Costs.** All of these devices tend to be very costly to purchase. One-end protection is the cheapest; it can be attained for a minimum of about $200 and a maximum of about $1,200 per port, depending on features and level of protection. However, the two-end devices are much more complex and costly; prices for complete systems, including terminal/user devices and host devices or software can run to as high as $3,000 per host/terminal link, depending on level of protection desired. These figures do not include installation or periodic maintenance and repair of equipment and software, both of which may be substantial.

- **Operating Costs.** There are a number of costs which may be incurred from device usage and operation, with the following being some of the most important. User costs due to reduced efficiency may be small incrementally, but can add up quickly. Most of the devices require the user to take additional steps and submit to delays in the process of signing on to the host. In addition, there can be significant labor costs from administering the security system, including password or user token management and related activities. For devices which use call-back, all the telephone usage tolls would be incurred by the host rather than the user, which may make it difficult to allocate these costs properly. On the other hand, call-back may permit the system administrators to standardize on a single, low-cost system such as WATS, and reduce overall telephone toll costs. Finally, some of the two-end devices make use of application software operating on the host computer instead of stand-alone hardware, thus incurring system overhead costs.

Management And Administration. In addition to direct and hidden cost factors, there are other potential drawbacks and problems in the use of dial-up security devices which may arise. These can be very significant, to the point of curtailing the usefulness of the devices. The following are some of the most important problems in usage that are often encountered.

- **Identifying Valid Users & Privileges.** A problem that is encountered when rigorous access control systems of any form are installed is to determine correct levels of user privilege. Many organizations simply do not have an easy way to correlate specific valid users with the specific computer systems and applications they should be permitted to access, given our modern and very complex communications environment. Any particular user may enter a system via dial-up, direct connect, local area network, wide area network, and so forth. Certain individuals may be authorized to access one computer system or application during normal work hours and use other systems at all hours. For large systems, it may take months to sort out this set of user access conditions, and they may be very difficult to keep current. This could substantially delay implementation of rigid dial-up access controls.

- **User Convenience (The Nuisance Factor).** An objection that often arises to improved security is that it tends to get in the way of valid users. As noted earlier, most types of dial-up security devices require user overhead in the forms of additional procedures to follow and insertion of delays in the connection process. There may be passwords to remember or special devices to carry and manipulate. For the infrequent user, the extra steps may be very confusing and frustrating.

- **Maintenance of Authenticators.** In operating dial-up security devices which use personal user authenticators or passwords, there is the major problem of administering a second password management system, separate from that used by the host computer. The procedures for assigning and changing these communications passwords should be rigorous, otherwise the real protection they can offer will be reduced. Usually, this means that more people will be needed to administer the system, which may significantly increase operating costs.

RECOMMENDATIONS

The overall challenge of improving dial-up security is no different from other types of security: determining the system's security needs, evaluating the present state of security, and selecting the optimal set of controls to raise that state to the desired level. This rules out selection of security hardware until it is determined that these devices are clearly justified. Normally, there are a number of other less-costly controls which should be considered before this justification can be accepted.

The following recommendations will aid the security administrator to act conservatively and still improve resistance against dial-up intruders.

1. System Security Administration. The first and most important step in improving dial-up security is to review and correct present security administration procedures. Weaknesses in this area are the single greatest cause of intruder penetrations. The focus here should be on ensuring:

- **Clear individual accountability,** and

- **Uniqueness in identification and authentication.**

In practice, the key points to stress are that vendor-supplied USERIDs should never be retained on the system, no user should share a USERID with another, no USERIDs should be assigned or retained without verified continuing clear need, passwords should be unique to each user and highly resistant to guessing, and passwords should be changed with increasing frequency as the level of security requirements rise.

2. Operating System Security Features. With the unfortunate exception of current microcomputer versions (e.g., PC-DOS), all operating systems have some features which can be used to improve their ability to counter dial-up penetration attempts. Often, however, system managers resist using these features because they tend to reduce the system's efficiency somewhat or impede user flexibility. On the other hand, installing external security equipment has similar effects and can be more costly. Some of the operating system features which are most useful are:

- **Use the system's journalling capability** to capture all security-related events, such as invalid attempts to log-on or execute restricted programs, creation of new user accounts, changing passwords, and the like.

- **Use the system's permission codes** (read/write/execute) to restrict access to files and programs. Set system defaults to make all programs and files private, which will require their owners to grant specific permissions to others as needed.

- **Use the system's ability to block invalid log-on attempts,** in all forms such as restricting number of attempts to a maximum of three and then timing-out ports for a short time.

3. Standard Equipment with Security Features. It makes good sense to purchase standard communications devices which have innate security features, rather than to obtain extra equipment solely to provide the same form of security protection. Various manufacturers now provide security-equipped devices such as: modems, protocol converters, multiplexers, port contenders and expanders, and data switches. The security features may include password tables, call-back, encryption, user authorization by time of day, port restrictions, and others.

4. Port Protection Devices (PPDs). If the host system's user identification and authentication procedures cannot be improved easily, and the system requires a moderate improvement in dial-up security, then PPDs may provide the added protection necessary. It is important to remember that PPD password management must be at least as strong as on the host, because the PPD's main function is to supplement that of the host.

Additional recommendations if PPDs are used:

- **Apply the call-back feature with caution.** It may not be needed or may induce additional weaknesses. Strong password procedures for the PPD are better in the long run.

- **Use maximum camouflage in PPD sign-on screens,** so that intruders cannot identify either the PPD or the host.

- **Use the PPD's logging features** and review the logged data frequently.

5. Terminal & User Authentication. If a greater degree of dial-up security is needed than provided by PPDs, then use terminal or user authentication devices where practical. For good routine security, user authentication tokens are adequate. For good user site identification, terminal authenticators are best. For higher levels of security, use the devices which provide both terminal and user authentication at the same time, such as terminal authenticators which have a slot for a user token.

6. Line Encryption. For the highest levels of dial-up security, use automatic line encryption devices which perform their own key management. A somewhat lower degree of security can be provided by encryptors which use manual key management.

7. The Last Word. To leaven all the above, the security administrator should keep in mind that computer systems exist to be used, and that their ready use is now required for carrying out the organization's mission. This implies that security and user productivity must be balanced in a rational way. It is important to avoid "user surliness" in installing additional communications security procedures or devices, by making the security features as transparent and easy to use as possible. Security is not mutually exclusive with dial-up connectivity.

ADDITIONAL READING

FIPS PUB 112, Standard on Password Usage, National Bureau of Standards, 1985.

FIPS PUB 113, Computer Data Authentication National Bureau of Standards, 1986.

Murray, William H., "Good Security Practices for Dial-Up Systems," Computer Security Journal, Fall-Winter, 1983, pp. 83-88.

Sandza, Richard, "Beware: Hackers at Play," Newsweek, September 5, 1984, pp. 42-48.

Steinauer, Dennis D., Security of Personal Computer Systems: A Management Guide, NBS Special Publication 500-120, January 1985.

Troy, Eugene F., "Thwarting the Hackers," Datamation, July 1, 1984, pp. 117-128.

Troy, Eugene F., "A Guide to Dial-Up Port Protection Products," Computer Security Newsletter, July/August 1984, p. 4.

Troy, Eugene F., Stuart W. Katzke, and Dennis D. Steinauer, "Technical Solutions to the Computer Security Intrusion Problem,", Workshop on Protection of Computer Systems and Software, National Science Foundation, October 22, 1984.

Troy, Eugene F., "Dial-Up Security Update," Proceedings of the 8th National Computer Security Conference, September 1985, pp. 124-132.

Troy, Eugene F., "Communications Security Equipment," Computer Security Newsletter, September/October 1985, p. 5.

Troy, Eugene F., Security for Dial-Up Lines, NBS Special Publication 500-137, May 1986.

AUTOMATED ANALYSIS OF COMPUTER SYSTEM AUDIT TRAILS FOR SECURITY PURPOSES

Lawrence R. Halme

John Van Horne

Sytek, Inc.
1945 Charleston Road
Mountain View, CA 94043

INTRODUCTION

Manual review of computer system audit trails is currently the only means available to monitor systems for security violations. Automatic tools are needed to assist computer system security officers in this task. This paper presents findings from an investigation into automating the analysis of existing audit trails for security violations through the use of pattern recognition techniques. The investigation included the analysis of actual audit data and simulated intrusion audit data. The results were applied to develop an automatic audit trail analysis tool. The investigation was performed for the Department of the Navy, Space and Naval Warfare Systems Command, under Contract Number N00039-85-C-0136. The results of this investigation demonstrate the success of this approach. The paper also discusses future directions for research.

BACKGROUND

Monitoring of computer system use for security violations will always be necessary. Even if we perfect the ability to design secure computer systems which we can trust, we can never fully trust their users. The problem of catching legitimate users who violate system security will remain a problem which can most effectively be addressed by security monitoring.

Currently, system security officers perform security monitoring of computer systems by manually reviewing the system audit trail. The only automated help available to them comes in the form of audit mechanisms capable of producing reports or data bases which store audit trail data. Consequently, there is a great need for more capable automatic tools to assist in this task. This need, and the lack of work being done to develop such tools, was pointed out by Marv Schaefer in his closing remarks to the Eighth National Computer Security Conference. Although in 1980, the James P. Anderson Co. produced an excellent discussion[1] of this problem, not much seems to have been done since then.

An automatic tool to assist in the task of security monitoring would require data about user activity on the system. Audit trails already provided by the system are one source of such data. They have the advantage that they are an economical and practical source, since their use would require the automatic monitoring tool only to interpret the data and not collect it. On the other hand, the disadvantage of the use of audit trails should be recognized. They may not have been originally intended for security purposes and may not contain enough security relevant material. The audit mechanism may not be secure itself, so that the audit data it produces may be of questionable integrity.

Whatever its source of monitoring data, an automatic tool can provide the most assistance to a system security officer by accurately identifying monitoring data which represent security violations. Perfect accuracy is likely to be very difficult to achieve, but a tool need not be perfectly accurate to be practical. Consider the two types of errors such a tool could make. It can identify as representing violations monitoring data which do not in fact represent violations, and it can fail to identify data which do indeed represent violations. The first type of error is by far more acceptable than the second type. If a tool could be developed which would not make errors of the second type, it could act as a reliable filter which takes in all system monitoring data and releases that data which it finds suspicious, including all data representing actual violations. The fewer errors of the first type the tool makes, that much more useful it would be.

In order to make decisions about which monitoring data represent violations, the tool will have knowledge about the system and its users. The favored approach is to have the tool understand normal patterns of system use for each user. Any monitoring data not falling into these patterns of normalcy would be considered suspicious and possibly representing a violation. Another approach would be for the tool to understand patterns of violations, and for it to report monitoring data which fit those patterns. While this approach may be valuable, it should not be relied upon solely since it is unlikely that all patterns of violations are known.

The main goal of the investigation discussed in this paper was to determine the potential of a tool whose source of monitoring data was an audit trail. A reasonable degree of success of such a tool which analyzed even a limited range of audit data would demonstrate that the approach would be more successful when applied to analyze more general monitoring data. More details can be found in the investigation's final report[4].

The approach taken in this investigation was to organize the audit trail data according to which session generated it. Sessions were to be classed as normal or intrusive based on patterns formed by their individual audit trail records. Functions of these fields, called features, were defined to characterize certain aspects of normal patterns for sessions. Parameters within the features would differ from user to user depending on individual system usage patterns. These parameters were assigned values according to audit data describing only normal activity in a process called training. The features were then tested for their ability to discriminate between normal and intrusive sessions. Features flag sessions which do not fit the pattern of normalcy which they describe. Successful features were combined to create for each user a user profile characterizing that user's normal system activity. An automatic audit analysis tool was developed using these user-profiles.

PROJECT SUMMARY

From a VAX-11/750 running UNIX we collected audit data that was analogous to audit data collected by general-purpose systems. (UNIX is a registered trademark of AT&T Bell Laboratories.) This was accomplished by altering the C shell command interpreter to collect additional event attributes over what the UNIX auditing/accounting facility normally collects. No attempt was made to construct a secure, tamper-proof auditing tool. The goal was simply to gather a representative audit trail. The data we collected was compared to the auditing information collected by commonly used systems such as SMF for IBM/MVS. Although only a fraction of what would be useful in characterizing usage patterns was collected, these fields are representative of what is likely to be collected by a commercial audit facility. The fields collected included the user-ID, commands issued by the user, the current directory, the port on which the user was logged in, internal file statistics, and internal process statistics. File statistics included owner-ID, size, and times of creation, last update, and last access. Process statistics included size of input and output, running times, and amount of memory used. Two sets of audit data, each representing one week of auditing, were collected and entered into an INGRES database.

In order to test which types of intrusions could be reasonably detected by auditing, we developed a set of twelve scenarios of abusive behavior. These scenarios included break-ins by hackers, legitimate users masquerading as other users, and legitimate users deliberately trying to subvert the system in a variety of ways. We elaborated each of these intrusion scenarios into a sequence of probable suspect actions. These sequences of actions were performed on the system, and the resulting audit data used as a test set.

Based on the fields of the audit data records, we defined features and prepared to test for their effectiveness. With few exceptions, each of the audit fields that we collected was used to define a feature. In most cases the features depended upon only one audit record field, yet there were features defined as combinations of more than one field. Thirty-two features were tested in all, among which were the time of day of use, the command usage, and directories and files accessed.

From the features, we defined a "certainty measure." The purpose of the certainty measure was to indicate the degree of suspicion of a session. If the certainty measure attributed higher values to sessions which more likely represent intrusions, it would direct the system security officer to the sessions which more urgently should be reviewed. The certainty measure would be regarded as effective if in testing, its values for the intrusion scenarios were much larger than its values for normal sessions.

Among the goals of testing was to determine which features were most effective at flagging the intrusion scenarios. Certainly, features which flagged no scenarios would not be considered useful for intrusion detection. A second goal of testing was to determine which features would also flag few normal sessions. Since the goal is to reduce the volume of audit data without eliminating data representing intrusions, features should flag as few normal sessions as possible. A third goal of testing was to determine how the features performed as a group, rather than individually. Because it would be unrealistic to expect that one feature would be sufficient to detect all intrusions, a user profile consisting of the best performing features would be needed in the audit analysis tool to detect as many intrusions as possible while also flagging as few normal sessions as possible. A final purpose of testing was to evaluate the certainty measure for individual features as well as for groups of features as an indicator of the likelihood that a session is an intrusion.

The tests performed involved the three sets of data: week one audit data, week two audit data, and the intrusion scenario audit data. Week one and week two data represented normal audit data. The tests were performed by using one of these normal sets for training the features, and then testing the features against the other normal set and the intrusion scenario data set. Thus, when week one was used for the training set, week two and the scenarios were the test sets. These tests were actually performed first. When week two was used for the training set, week one and the scenarios were the test sets.

In the tests using week one for training, nearly every feature flagged at least one intrusion scenario. Therefore, the main measure for the performance of the features individually was how few normal sessions they flagged. Twelve features which flagged no more than fifteen percent of the week two sessions were chosen for the tests with week two as the training set.

These tests confirmed that these features performed adequately. It is especially noteworthy that in both cases of training, the user profile formed from these twelve features flagged all intrusion scenarios, and when trained with week one, it flagged only 40.9% of the normal sessions. We considered this quite good, since it was the first test of this unoptimized user profile. The performance of the certainty measure for an intrusion scenario was twice that of a normal session.

The features which appeared to be effective fall into four categories. There are specific reference features, file statistic features, features based on process statistics, and a command usage pattern feature.

The three specific reference features spotted references to commands or files which were used by one of the intrusion scenarios to subvert system security. They were included in the tests because it was believed that these references occurred seldom in normal use of the system. The test of the effectiveness of these three features was whether they would flag an unacceptably high number of normal sessions. The results were very good, with very few normal sessions being flagged. This seemed to be a small price to pay for capturing the intrusive sessions which also have this feature.

The features defined by accessed file statistics which were among the effective features were defined by the device on which the file resided, the size of the file, and the user-ID and group-ID of the owner of the file. The resident device of the file referred to which disk drive the file was on. Most users showed little variation in which disk drives they accessed, as demonstrated by how few normal sessions this feature flagged. The file size feature identified the intrusion scenario in which a user tries to bring down the system by creating large files and occupying all available disk space. It also flagged a scenario in which a hacker breaks into the system. As expected, the features defined from the user-ID and group-ID of the owner of the file accessed identified scenarios that included browsing.

The effective features defined from process statistics dealt with time of use, timing of the process, and memory use. Time of use features effectively caught scenarios of hackers breaking in at night or over the weekend, as well as legitimate users logging in at unusual times to abuse the system in some way. The measure of timing of the process which was most effective was the CPU time of the user programs (as opposed to the system programs) associated with the process. In this way, excessive processing was flagged. The memory use feature recorded a range for the maximum memory used by the process. Since users have little direct control over memory use, it was surprising that this feature was so effective. Apparently, it is an indication of intensive processing.

The best performing command usage pattern feature was one which recorded for each command a range of the minimum and maximum percentage of the session time spent in the command. Time was measured by CPU time. This was one of several tested features designed to measure how much each command is used. It was interesting to learn that CPU time is a better measure than real time.

The other features tested failed because they were unadaptable, because of the poor quality of the audit fields they were based on, or because they were simply poor indicators of normal activity. For instance, the file name and the current directory were the basis for two features which performed badly. Since files and directories are created by users rather often, numerous false flaggings occurred. In order to be useful, features built on these fields should be able to adapt to this dynamic situation. They need to be able to learn when a new file or directory is created so that it can be added to the list.

A pattern classification tool was developed incorporating the user profiles based on these twelve most effective features. Besides increasing the ease and efficiency of testing whether certain features of the audit trail database are useful discriminators of intrusions, this tool can be considered a prototypical audit analysis tool. It is essentially a user-interface designed to provide a system security officer with the capabilities needed to use the audit data base for security purposes. The user can train the user profiles, query which sessions within the data base are flagged by the user profile or by any subset of the features in the user profile, and view sessions satisfying a property specified by the user, such as sessions with certain measure values higher than a certain threshold.

FUTURE RESEARCH DIRECTIONS

The results of this investigation demonstrate a successful approach to the automated detection of intrusions from audit trails. The basis of this conclusion is the performance of the prototypical audit trail analysis tool developed and tested in this project. While this tool could be adapted for any system similar to the one for which it was developed, the approach could be applied to virtually any system. Further research is needed to take full advantage of the results of this project and to develop a practical tool.

One area of future research must be the development of features and certainty measures which are more effective at discriminating between normal and intrusive audit data. The application of expert system technology to this discrimination should be investigated. It is also important to determine what other monitoring data, not normally contained in audit trails, would be useful. Selecting data fields which are common to all systems meeting a particular classification as defined in the "DoD Trusted Computer System Evaluation Criteria" would make a more generally applicable tool. Another area requiring further investigation should be the determination of the amount of data needed to train features most effectively. The performance of a feature cannot be accurately judged if it is inadequately trained. Finally, a large and very significant questions should be how to apply these results to other computing environments, such as DBMSs and networks. Specialized environments offer fertile ground for progress in this area. With their narrower capabilities, these systems would have narrower definitions of normal use which could be more easily characterized.

BIBLIOGRAPHY

[1] James P. Anderson Co. "Computer Security Threat Monitoring and Surveillance," Fort Washington, Pennsylvania, April 15, 1980.

[2] Dorothy Denning and Peter G. Neumann. "Requirements and Model for IDES -- A Real-Time Intrusion-Detection Expert System," SRI international, Menlo Park, California, August, 1985.

[3] Lawrence R. Halme, Teresa F. Lunt, John Van Horne "Results of an Automated

Analysis of a Computer System Audit Trail." Proceedings of the Second Annual AFCEA Physical & Electronic Security Symposium and Exposition, Philadelphia, Pennsylvania, August, 1986.

[4] Sytek, Inc. "Analysis of Computer System Audit Trails -- Final Report," Sytek Technical Report TR-86007, Mountain View, California, May 30, 1986.

MANAGING EXPOSURE TO POTENTIALLY MALICIOUS PROGRAMS *

Maria M. Pozzo
Terence E. Gray

Computer Science Department
University of California, Los Angeles

Abstract

In a resource-sharing environment, existing security mechanisms are often inadequate in defending a system against programs that contain malicious code such as Trojan horses and computer viruses. Approaches to reducing potential damage caused by such programs include: limited sharing, dynamic auditing, detection of modified programs, and decreased exposure to high-risk software. A risk management mechanism is proposed that allows administrative classification of software based on the credibility of its origin, and permits individual users to specify which classes of software they wish to be exposed to. The goal is to give users a way to avoid unwitting use of high-risk software. This Risk Management Scheme is not intended to be a complete solution to the problem of programs that contain malicious code; rather, it is intended to complement the authors' previous work in the area of computer virus containment.

Introduction

When invoking a program, a user has expectations about its behavior based on documentation, experience, and possibly the source code itself; however, the actions of that program are only indirectly visible to the user, if at all[1]. Thus, it is possible that the program contains some hidden function that could have harmful side-effects such as those caused by Trojan horses and computer viruses[2]. Ideally, a computer system should contain automatic mechanisms to prevent introduction of such programs into the system; however, existing preventative mechanisms are often unsuitable or inadequate. Moreover, a prudent system administrator would not place complete confidence in the effectiveness of any single control.

Other than preventing the introduction of suspicious programs into a system, what can be done to avoid damage? Protecting users from malicious programs can be accomplished in several ways:

- by restricting users from sharing programs, thus isolating the user from potentially malicious programs. Unfortunately, this isolationist approach is not be acceptable if users are to benefit from each others work[2].
- by auditing the behavior of programs during execution, and reporting suspicious actions to the user and/or system administrator. If one could infer malicious activity with complete certainty, then the system could prevent further execution, but the chance of halting execution erroneously is very high. Also, with an auditing approach a user might be inundated with records of inocuous actions.
- by determining that a program is potentially malicious prior to run-time, and preventing its execution. While it is not generally possible to statically analyze an executable and infer with confidence its proclivity for damage, it is possible to detect modification of executables since their installation[3], a technique that appears to be promising for containing the spread of computer viruses.
- by applying procedural controls such as physical mechanisms (e.g., guards, restricted areas of operation), configuration management policies, standard development practices, etc. Many of these procedural controls should be in existence in all systems, and provide no additional protection against malicious activity.
- by classifying executables according to the likelihood that they contain malicious code, and giving users a way to avoid unwitting use of high-risk software.

The last approach is the subject of this paper. A Risk Management Scheme is proposed that provides for administrative classification of software based on the likelihood that an executable is free of malicious code and also permits each user to specify which classes of software can be executed on his or her behalf.

Software on a system can come from many places and there *are* differences in the credibility of various individuals and organizations who develop software for a system. These differences can be reflected by classifying the software based on the credibility of it's origin as determined by the system administrator. Thus, the likelihood that an executable is free of malicious code is determined by the credibility of its origin. Users can then manage their vulnerability by choosing a risk level at which they are willing to operate, thus controlling their exposure to potentially malicious programs.

* This research was supported in part by the NSF Coordinated Experimental Research program under grant NSF/MCS 8121696 and by the IBM Corporation under contract D850915.

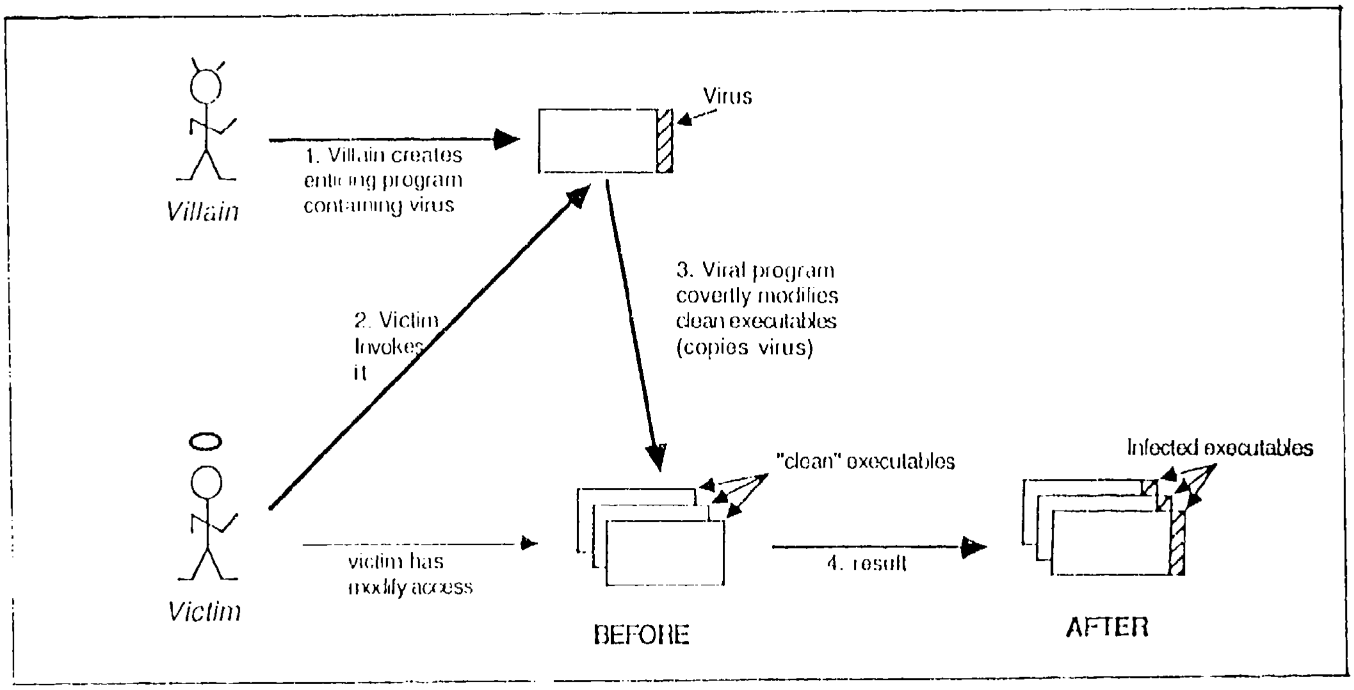


Figure 1: The Process of Infection

The next section provides background material on the types of malicious programs and the damage they can cause as well as some traditional protection models. A more detailed look at the problem is presented in this section. The Risk Management Scheme is then discussed followed by consideration of its strengths and weaknesses. The issues discussed in this paper are part of an on-going effort. Our long-range goal is to develop a complementary set of independent mechanisms for protection against computer viruses and other malicious programs.

Background

Not all program side-effects or hidden functionality are bad. This discussion, however, is concerned with hidden code that is deliberately inserted by unscrupulous individuals, with the intention of causing malicious side-effects[4,5]. A Trojan horse* lures unsuspecting users into executing it by pretending to be nothing more than a useful or interesting program[6], while in reality it contains additional functions intended to "...gain unauthorized access to the system or to [cause a] ...malicious side effect"[5]. The difference between a computer virus and a Trojan horse is that a virus "...can 'infect' other programs by modifying them to include, a possibly evolved, copy of itself."[2] Trojan horses and computer viruses are particulary insidious because they operate through legitmate access paths, taking advantage of normal access rights belonging to the user. Figure 1 depicts this operation in the case of a computer virus.

The system's file space contains several "clean" executables to which the victim possesses modify access. The villain creates an executable that performs a function designed to entice unsuspecting victims to invoke it. Embedded in the executable is a piece of clandestine code that is a virus. When the program is executed, the hidden viral code is executed in addition to the program's normal service. The victim, however, only sees the normal service, and therefore, does not detect the presence of malicious activity. The virus program, when executed by the victim, typically carries the victim's access rights and, therefore, has modify access to all of the victim's executables as well as any other programs for which the victim has legitimate modify access. The virus copies itself to the victim's uninfected executables. Further, when any other user (with appropriate access rights) invokes one of the infected programs, the virus spreads to that user's executables and so on. In addition to its spreading property, the virus may contain a Trojan horse intended to cause damage of some kind.

Several properties of typical computer systems lead to an environment in which malicious programs can wreak havoc: the need for program sharing[5], the difficulty in confining programs*, and the fact that existing discretionary access control (DAC) mechanisms are fundamentally flawed with respect to limiting Trojan horses[1] or computer viruses.

* "The Trojan horse works much like the original wooden statue that the Greeks presented at the walls of Troy--it is an attractive or innocent-looking structure (in this case, a program) that contains a hidden trick, a trick in the form of buried programming code that can give a hacker surreptitious entry to the system that unknowingly invites the Trojan Horse within its figurative walls. The Trojan horse is very simple in theory, but also very effective when it works. The program that is written or modified to be a Trojan horse is designed to achieve two major goals: first, it tries to look very innocent and tempting to run, and second, it has within itself a few high-security tasks to try."[7]

* A program that cannot retain or leak any of its proprietary information to a third party is confined[10].

Several mechanisms exist for limiting the amount of sharing such as the security and integrity policies[8,9], and flow lists or flow distance policies[7]. However, to the extent that these mechanisms permit any sharing, the damage caused by Trojan horses and viruses cannot be eliminated since their malicious activity is conducted via legitimate access paths due to the fundamental flaw in DAC. Some work has been done in the area of program confinement[10,11] and towards solving the DAC problem[12]. Because of the difficulty in preventing and detecting malicious activity, a scheme is proposed here that can be implemented with

The system administrator* assigns software a credibility value which identifies the likelihood that the software contains malicious code. In general, this value is based on the origin of the software. Credibility values range from zero to N, where software with the lowest credibility has the value of zero and software with the highest credibility on the system has the highest value. Software that is formally verified, so that the possibility of it containing malicious code is small, is always assigned the highest value. The number of credibility values is determined by the system administrator and can be one. For example, in an environment where security is of primary concern such as a military installation, a system may be restricted to only verified software. An environment where security is of less concern, is unlikely to have any formally verified software. But, since differences exist in the credibility of the various sources of executables, the system administrator can choose some number of credibility values to reflect the classes of software on the system. Figure 2 depicts a possible configuration for credibility values.

Origin	Credibility	User's Risk
User Files	0 - Lowest	0 - Highest Risk
User Contributed S/W	1	.
S/W from Bulletin Board	2	.
S/W from System Staff	3	.
Commercial Application S/W	4	.
S/W from OS Vendor	5 - Highest	5 - Lowest Risk

Figure 2. Credibility Value and Risk Level

Per-User Risk Management

Risk levels specify what classes of software can be executed for a user. They correspond inversely to credibility values. If the user's risk level is set to the highest credibility value on the system, the risk of damage to that user is the lowest possible. On the other hand, the greatest risk is taken when the user specifies a risk level of zero.

When a user logs in, a risk level is established for the session. This risk level can be determined in two ways. The first way is for the user to specify the desired risk level as an argument to the login command (e.g., login Joe -session_risk 3). The second way is to assume the default risk level for that user. Initially, the default risk level for all users is the highest credibility value on the system. The user can reset this default risk level by specifying the desired default as an argument to the login command (e.g., login Joe -default_risk 2). The user need only set this once and it remains in effect until it is explicitly reset by the user. Thus, assuming the default risk level as the risk level for the session requires no explicit action on the user's part once it is set. Once the risk level for a session is established, any processes that are spawned inherit the risk level of the parent, restricting children to running software of the same credibility value or higher. The only way for a user to override the risk level for a particular session is via the RUN-UNTRUSTED command which takes one executable program as an argument. This program can have a credibility value less than the risk level. The duration of this exception is the execution of the program supplied as an argument. The objective of the "RUN-UNTRUSTED" command is to make execution of high-risk programs explicit, but not too inconvenient.

* The system administrator is considered to be one or more individuals, trusted not to compromise the security or integrity of the system.

Credibility Value	Execution Mode	User's Risk Level
0	RUN-UNTRUSTED	
1	RUN-UNTRUSTED	
2	RUN-UNTRUSTED	
3	normal	
4	normal	Risk Level 3
5	normal	

Figure 3. User's Risk Level

As an example, Figure 3 shows five possible credibility values for software, where the existence of malicious code in software with a value of 5 is unlikely and in software with a value of 0 is most likely. The initial default for the user is the ability to run software with a value of 5 only, unless the user explicitly logs in at a lower risk level or resets the default risk level. If the user chooses to establish a session with a risk level of 3, software with values of 0, 1, and 2 cannot be run without using the RUN-UNTRUSTED command. Of course, the user has increased the potential risk of exposure to malicious activity.

System Configuration

Once a credibility value has been assigned to software, the information must be conveyed to the run-time environment. This can be accomplished in several ways. The first approach is to store the credibility value as part of the executable, comparing the value with the user's risk level prior to permitting execution. This approach requires that the executable be protected from modification to ensure the integrity of the credibility value. A second approach is to keep a list of all executable software in the system and the associated credibility values. When a user executes a program, the run-time environment searches the list for the program's credibility value and compares it with the user's risk level before allowing execution. Such a list must be protected from illicit modification. This approach may not be practical depending on the time it takes to complete the search. A third approach is to group software of the same credibility value in the same place in secondary storage, and maintain a short, protected list mapping credibility values to each file group. Software of the same credibility value could be stored in the same directory, in the same filesystem* , or some other mechanism used to partition software. The list identifying each partition and the associated credibility value is then short enough to avoid performance problems, but must still be protected from modification by anyone except the system administrator. Figure 4 shows possible credibility values for software grouped using Unix** directories as the partitions.

As the number of credibility values is determined by the system administration, so is the granularity of the partitions. For example, one system might partition all vendor software into one partition with the same credibility value while another system might have separate partitions for IBM, DEC and AT&T software, each with a different credibility value.

Origin	Credibility	Partition
User Files	0 Lowest	/usr
User Contributed S/W	1	/usr/flakey
Bulletin Board S/W	1	/usr/net
Commercial S/W	2	/usr/bin2
S/W from System Staff	3	/usr/local
Commercial S/W	3	/usr/bin
Verified S/W	4	/usr/ver
S/W from OS Vendor	5 - Highest	/bin

Figure 4. Partitioning Software of Different Credibility Values

If an individual program becomes suspected of containing malicious code, perhaps based on reports from other installations, it can be moved to a different directory of appropriate credibility value. However, one disadvantage of associating a credibility value with separate directories or filesystems is that the full name of a program may be embedded in other programs or scripts; thus moving a program to a different directory having the desired credibility level is essentially a name change for that program, and may cause existing scripts to break. This observation argues in favor of assigning credibility values to individual programs, even though to do so is more administratively demanding. A combined approach that allows easy assignment of credibility levels to collections of programs, but provides for individual exceptions may be the winning strategy.

Strength And Weaknesses

The major strength of this concept is that it makes the user aware of the potential risk in executing certain programs. Disallowing execution of software below the user's risk level brings to the user's attention something that is potentially dangerous, in other words, executing such a program does not meet the system's standard criteria for program execution at the user's current risk level. Forcing the user to invoke the RUN-UNTRUSTED command in order to perform such an action lets the user know that this boundary is about be crossed. The use of the RUN-UNTRUSTED command does, however, intrude on normal user operation. In choosing a default risk level, typical users will try to ensure that the majority of commands they invoke will *not* require use of the RUN-UNTRUSTED command. This tendency contradicts the atmosphere of safety this mechanism is attempting to create. To retain this atmosphere, but still allow the user as much flexibility as possible, the system administrator can specify, on a user by user basis, the minimum risk level at which a particular user is ever allowed to operate. A system programmer or operator, for example, may be restricted to a higher minimum risk level than a normal user. This means that the default risk level can never be set lower than the minimum risk level for the user. Further, when the RUN-UNTRUSTED command is invoked, programs executed below the risk level of the session can never have credibility value less than the minimum risk level for the user.

Determining how to classify software from different origins is a subjective decision. The system administrator must determine this value based on past experience with the supplier, supplier reputation, and any other available measures. For example, vendor XYZ may have a good reputation in the field, and it is considered unlikely that any software they supply will contain malicious code. Software on a network bulletin board, on the other hand, has been know to contain Trojan horses and computer viruses. If the system has been partitioned as in Figure 4, software from a trusted vendor XYZ would be assigned a credibility value of 3, and placed in /usr/bin, whereas programs from a vendor not known for their configuration management might be assigned a credibility value of 2 and placed in /usr/bin2. Software from the bulletin board would be assigned a value of 1, and be installed in /usr/net or /usr/flakey. A set of guidelines should be created to maintain consistency. The credibility value is a subjective indicator, and thus, a weak point in the overall concept. This is not a fatal weakness, however, since the mechanism is intended to be a warning system. The point is to make visible any action which carries potentially unacceptable risk.

Many systems allow the user to specify where the operating system should look to find commands invoked by the user. For example, the user may wish the operating system to first look in the user's directory and if the desired command is not found, to then look in the experimental system libraries, and if it's still not found to look in the normal system libraries. This process is often called name resolution, and the means by which the user specifies the order of the locations to search is often called the user's search path. A search path is generally in effect for the duration of a user session and is specified by the user as part of the session start up. If the search path includes at least one directory that contains malicious programs, such as a user-contributed software library, or the user's "current working directory", then vulnerability is high. For example, suppose a malicious program is given the same name as some legitimate program. The perpetrator carefully places the malicious program in a directory that is searched before the one containing the legitimate version. The result is that the user executes the malicious program instead of the intended legitimate one. With the proposed system, if a name resolves to a program with a credibility value less than the risk level established for the session, execution is prohibited unless the RUN-UNTRUSTED command has been invoked and the credibility value is not less

* In Unix**, a filesystem contains a hierarchical structure of directories and files and corresponds to a partition of a disk. Each filesystem is represented internally by a unique number.

** Unix is a trademark of AT&T Information Systems than the minimum allowed for the user. Thus, the set of potentially damaging programs is reduced to those possessing credibility levels equal or greater to the user's chosen risk level, plus those programs executed explicitly via the RUN-UNTRUSTED command (further restricted by the user's minimum allowable risk level).

A set of guidelines should also be available to help the user determine the most appropriate risk level at which to operate depending on the such factors as: the sensitivity of the information the user is working on (e.g., proprietary, company confidential, burn-before reading, the new rogue game), the type of information to which the user has access (and to which any programs run by the user will also have access), the type of environment the user is working in (e.g., development, operator, maintenance), an so on.

Second only in importance to the usefulness of the *concept* is the feasibility of any candidate implementation. When considering implemention of the proposed Risk Management Scheme, dependencies on the underlying operating system must be identified. There are six critical aspects to the mechanism:

- the installation of software into directories with high credibility values;
- the integrity of executables after their installation;
- the credibility value associated with a particular program or partition;
- the files or data structures storing risk level information;
- the illicit use of the programs that implement the mechanism;
- the integrity of the operating system kernel.

Of primary concern is how easy it is to subvert the mechanism; for example, how easy it is for a perpetrator to get malicious code installed on a system with a high credibility value, or to change a user's risk level.

It is essential that installation of the software be restricted to the system administrator, otherwise, dangerous software can masquerade at a high credibility value. For example, in the configuration shown in Figure 4, if a perpetrator could install software into the /bin directory it would assume a credibility value of 5, the highest on the system. Thus, failure to restrict installation privileges renders this mechanism useless.

Since the effectiveness of the system also depends on preserving the integrity of the executables themselves, this scheme might be combined with an encryption mechanism as proposed in [3]. Allowing an executable to be modified after it has been assigned a credibility value and installed in a partition invites insertion of a Trojan horse or computer virus. Then, the assigned credibility value will no longer reflect the possibility that the software contains malicious code.

Protecting the credibility value associated with a particular program or partition was discussed in the System Configuration section. Risk level information falls into two categories: the default risk level and the minimum risk level which are associated with a user; and the process risk level which is associated with a user's process. The default risk level and the minimum risk level can be treated as part of the user's authentication information, as indicated previously. These items can then be protected in the same manner as user passwords. Protection of the per process risk level is dependent on the underlying system. If the underlying system is secure, all access to the process risk level can be mediated by the Trusted Computing Base (TCB)[15]. If the underlying system is not secure, alternative measures must be taken to protect the process risk level. This issue is addressed in [16].

In addition to being protected from illicit modification, programs that implement the Risk Management Scheme must also be protected from illicit use. Routines that set the minimum risk level for a user and set the credibility value for programs should be restricted to the system administrator. In an unsecure system, these operations can be protected by performing them in a system stand-alone mode, ensuring that they are not available during normal user operation. In a secure environment, they would be considered privileged operations, and part of the TCB. Setting of the default risk level by the user, if implemented as part of the login process, can be treated in a similar fashion to authenticating user passwords. Setting of the process risk level is accomplished at process creation time. If the system is secure, authentication and process creation would be considered trusted operations, and would be part of the TCB. Protecting this mechanism in an untrusted computing environment is addressed in [16].

Conclusions

We have proposed a mechanism that allows users to manage their risk of executing potentially malicious programs. The underlying premise of this mechanism is that useful distinctions can be made about software based on its origin. This mechanism is not a complete solution to the problem of malicious programs; it is intended to complement preventative mechanisms that currently exist as well as those described in our previous work[3].

Our future plans are to examine a prototype implementation of the Risk Management Scheme in order to fully investigate all of the implementation dependencies.

References

[1] Boebert, W.E. and Kain, R.Y., "Secure Computing: The Secure ADA Target Approach", Honeywell Secure Technology Center, Minneapolis, MN. 1985.

[2] Cohen, F., "Computer Viruses", Proceedings of the 7th DOD/NBS Computer Security Conference, September 1984, pp 240-263.

[3] Pozzo, M.M., Gray, T.E., "Detecting Modification of Executables Using Encryption", 1986, (unpublished).

[4] Lackey, R.D., "Penetration of Computer Systems: An Overview", Honeywell Computer Journal: 81-85, Sept. 1974.

[5] Denning, D.E., "Cryptography and Data Security", Addison Wesley Publishing Co., Reading, Ma, 1982.

[6] Anderson, J.P., "Computer Security Technology Planning Study", USAF Electronic Systems Division, Bedford, Ma., Oct. 1972, ESD-TR-73-51.

[7] Landreth, B., "Out of the Inner Circle: A Hacker's Guide to Computer Security.". Microsoft Press, Bellevue, WA, 1985.

[8] Bell, D.E., and LaPadula, L.J., "Secure Computer System: Unified Exposition and Multics Interpretation". MITRE Technical Report, MTR-2997, July 1975.

[9] Biba, K.J., "Integrity Considerations for Secure Computer Systems". MITRE Technical Report, MTR-3153, June 1975.

[10] Lampson, B.W., "A Note on the Confinement Problem", Communications of the ACM 16 (10):613-615, Oct, 1973.

[11] Lipner, S.B., "Non-Discretionary Controls for Commercial Applications", Proceedings of the 1982 Symposium on Security and Privacy, April 26-28, 1982, Oakland, CA., pp 2-10.

[12] Boebert, W.E., and Ferguson, C.T., "A Partial Solution to the Discretionary Trojan Horse Problem". Honeywell Secure Technology Center, Minneapolis, MN.

[13] Lipner, S.B., "A Comment on the Confinement Problem", The MITRE Corporation, MTP-167, Nov. 1975.

[14] Bourne, S.E., "The UNIX System", International Computer Science Series. Addison-Wesley Publishing Company, 1983.

[15] DoD Computer Security Center, "Department of Defense Trusted Computer System Evaluation Criteria", DoD, CSC-STD-001-83, 1983.

[16] Pozzo, M.M., Gray, T.E., "Computer Virus Containment in Untrusted Computing Environments", IFIPs, Dec 1986.

SECURITY ON UNCLASSIFIED SENSITIVE COMPUTER SYSTEMS

Hal Feinstein
The MITRE Corporation
7525 Colshire Drive
McLean, VA 22102

INTRODUCTION

This paper deals with some of the security issues facing unclassified sensitive computer systems that are operated by the civil agencies of the Federal Government.

Computer security has taken two directions; the first is prompted by the military and principally designed to serve the military need for secure command and control and the handling of classified information. This has centered primarily on applying trusted software to solve the multi-level security problem.

The second direction of computer security has been aimed at the unclassified, sensitive systems, such as are used by the civilian agencies which deals primarily with domestic matters. Here, emphasis has been on risk reduction and management within limited resources.

There has been considerable desire to share appropriate technology developed by the military with the civilian sector. Problems of doctrine and environment require redefinition for the civilian community which is often overlooked. This paper addresses some of these issues and suggests interim approaches where appropriate technology will not be available in the near timeframe.

There has been a tendency to erroneously place a civilian agency's unclassified information systems within a military continuum of classifications, relegating it to the lowest rung of protection chiefly because it bears the formal unclassified designation.

The military classification hierarchy is based on national security sensitivity. Thus, because national security considerations are not commonly involved with domestic data, the security of computer systems handling such data is not addressed from the military security viewpoint. Thus, without a formal civilian sensitivity ranking system, it is difficult for system managers to ascertain what level of protection is required within the trusted computing base.

A civilian agency information system has certain distinctive characteristics. They form a resource-access boundary and, in many cases, may represent a semi-autonomous structure. Commonly, an information system may be based upon a commercial data base management system or transaction monitor that has control over files, user terminals, and execution within its confines. Often, it is administered under project auspices instead of a data center manager.

Civilian agency information systems may span one or more computers and contain data of the highest sensitivity. Yet, because of economic and organisational constraints, it may be forced to operate next to or share computing resources with uncontrolled or minimally controlled systems.

The information systems discussed in this paper operate below the B-class of DOD trusted computer base (TCB)[1]. Thus, there can be less reliance on the TCB and more on designing assurance into each information system. The assumption that the C2 (very sensitive) level is sufficient on these systems has not been clearly demonstrated.

Perhaps a major, but often overlooked problem, is the cultural differences between the military and civilian domains. The emphasis and assumptions differ vastly between these two groups and has resulted in confusion. One symptom has been the inappropriate application of the military classification hierarchy mentioned above. In general, there is a view that the unclassified civilian agencies fit neatly with the framework already developed for the military and, by implication, the guidelines for using the TCB. In opposition to this is the fact that the civilian sector has evolved under a different cultural framework and, hence, what is needed is a new conceptualization of security and not a simple transfer of doctrine as some would suggest.

This paper describes the experience the author has had in examining a number of civilian agency systems and some of the issues which make direct application of the TCB and military guidelines somewhat inappropriate.

This paper is divided into eight sections. Section 2 reviews the environment of the civilian agencies from a view of their mission and organizational framework. This is essentially an era of shrinking resources, the Gramm-Rudgman amendment, and growing workloads. It is also an era in which the traditional centralization of the computer center is giving way to decentralization. Section 3 contains a discussion of password usage. More than any other protection mechanism, password type, method of issue, and security are an indicator to the individual style of their associated information system and, often times, their form and usage is viewed as a management prerogative.

Section 4 contains an overview of the method used to construct a security model. This model combines the information to be protected with the threat factors to yield a mapping to the TCB. This model is important because it reviews a central difference between military and civilian cultures--threat and countermeasures.

Section 5 contains a proposed mapping between asset values, access clearances, and required software trust. This model is proposed as a basis upon which civilian agencies might build.

In Section 6, the C2 class of the TCB is reviewed and compared against the threat model for sufficiency. This necessarily centers on the lower levels of the TCB below Class B; however, argument for a higher level is presented, especially for the very sensitive information on shared hosts.

Section 7 briefly reviews the issues associated with file encryption as a viable tool for civilian agency sensitive systems. Lastly, the paper concludes by reviewing the various issues presented.

ENVIRONMENT

This section reviews some of the factors which are germane to civilian sector systems and affect the operation of unclassified but sensitive information systems in that sector. The environment is characterized by the need to process a large amount of data of varying degree of sensitivity within the same system. With the move toward greater productivity and shrinking resources, data processing is expected to help streamline existing agency missions. Yet, in some cases, the rate at which the workload has increased outstrips the resources available to civilian sector computer centers.

There are several factors that shape many of the decisions made in civilian sector systems and in some ways also shape the acceptable security approaches. The priority of mission, from the standpoint of many civilian sector managers, is the need to process, in a timely fashion, very large amounts of non-sensitive information. Commonly, this forms the essential mission of the agency and, hence, the chain of command within the agency views a data processing operation as successful if it fulfills this demand. Thus, funds and resources are commonly devoted to servicing this goal first.

The civilian sector agencies commonly depend on commercially-available operating systems and system software. Modifications and special enhancements are typically added to improve the speed and reduce operational problems. System programming talent within these agencies is typically dedicated full-time to maintaining the service capability of the computing systems.

The appearance on the market of commercial access control packages have greatly enhanced security in these centers by providing a package of security tools not requiring development but only installation.

The second major factor which conditions unclassified sensitive systems is that there is no consistent, community-wide classification system in use across all civilian sector agencies. The military or national security classification hierarchy deals specifically with information affecting the national security. Some information handled by the civilian agencies does affect the national security and this information, when originating within classified programs, bears the proper national security or military designations.

There is no parallel classification scheme and handling protocol available to civilian sector managers. Some departments have instituted special handling categories which apply only to their department. Commonly, they designate a two-level approach, designating a restricted class of information which must be specially handled. In some cases, this special handling level resembles a mixture of "official use only" and "confidential." Often times, this single special level is meaningless when applied to all the agency's sensitive data because it does not provide the handler with a clear indication of the information's true sensitivity. Hence, other agencies which may act as temporary custodians of the information for purposes of analysis or storage may not routinely recognize or enforce the information's true protection requirements.

Most special handling caveats are department or agency specific. When information is transferred across department or agency boundaries, there is often an uneven and ad hoc approach to information protection. This is, in part, the result of the assumptions producers make about how the information is being protected and how serious the recipient consumer or custodian is in providing the protection. It seems that this often unverbalized assumption presents one of the real dangers in the unclassified sensitive sector because the producer of the information may be only vaguely aware of the exposures present in a custodian's or consumer's computer system.

There have been limited efforts to create a set of sensitivity classes which would be recognized government-wide. The National Telecommunication and Information System Security Committee (NTISSC) staff is attempting to define a meaning for the term "sensitive"; however, this effort seems confined to designating unclassified data which is important to the national security as opposed to information of domestic interest.

Defining government-wide sensitivity levels for domestic data would go a long way toward eliminating the confusion and variability in data protection commonly seen today. It would vastly improve the way in which data protection planning is currently approached by providing a set of well-defined planning targets and a consistent measurement of provided protection.

A third factor is the need to make use of systems currently in place and, in addition, the need to share resources. A sensitive information system often executes, side-by-side, with non-sensitive and perhaps even uncontrolled systems. This is a result of the need for sharing of mainframes since there is an economy of scale trade-off typically used in planning many large computer systems.

It is not uncommon to see jobs of various security levels, both online systems and batch jobs, executing on the same mainframe. This is an example of multi-sensitivity (similar to multi-level but with unclassified information) operations run under non-trusted software. Multi-sensitivity operations on untrusted software carry with it a risk which is difficult to estimate. In an extreme case which is, however, not hypothetical, very sensitive information which is available only to a restricted set of specifically "cleared" individuals runs side-by-side with non-sensitive information systems which have minimally controlled access.

Thus, how much confidence should be placed in the commercial operating system, disk access, terminal handler, transaction monitor, and data base to ensure that unwanted intrusions are prevented? There are two additional dimensions to this problem--separating different groups of cleared individuals within the information system and the old-new problem.

Gaining administrative clearance to access an information system usually entails two separate and different accesses. The first allows the user to sign-on to the online system itself, while the second permits him to access certain sensitive files within the information system. This latter access might be termed "file access" since this is the mechanism where access control is typically enforced.

We have already asked the question concerning the ability of the information system to prevent outside users from accessing it; namely, access by users who have not been granted administrative access to the information system at all.

The second question we pose is can an information system prevent users who have access rights to it from accessing files to which they do not have a right? This question is central to secure software efforts and can probably be answered in the affirmative. Demonstration on a number of trusted systems, such as the Honeywell SCOMP and the various secure UNIX efforts, is possible. However, how should it be answered for a traditional commercial transaction processing package and operating system?

The second dimension of the problem is the old/new issue which arises from attempting to modify and extend old information systems. Often these systems were designed with elementary or inadequate security considerations. These systems have been systematically extended over the years resulting in a culmination of "loosely coupled" software subsystems. Security control is often ad hoc, or easily bypassed; few, if any, sophisticated penetration analyses have been performed and only rudimentary audit trails exist when they exist. In some cases, passwords are stored in plain text, something encountered more commonly than would be expected; in other cases, "homemade" password and data encryption algorithms are used.

Comparatively, some newer sensitive information systems currently under development have well-designed security features built into the lower level of the information system. This lower level is a form of executive process for the information system preforming and, hence, mediating access to sensitive files, devices, and transactions. Each new application within the information system will find that security mechanisms, such as access control, auditing, authentication, and transaction control, are built in as primitive, low-level operations. The lower level security "base" is maintained by an experienced and trusted system programmer, while the complicated and extensive applications code might be contracted out to a software development contractor.

Agencies responsible for design of the newer information systems commonly include security as one of the primary attributes needed by their system from the beginning of the planning process. In addition, they understand what modern software security approaches are available and see to it that they were included in the system design. Often, a central access matrix or access list is used to allow easy maintenance of user privileges. These information systems tend to have some of the best security characteristics encountered for that environment.

Comparatively, some information system design groups take the approach that security would be added on after the information system was implemented. These groups tended to view security as primarily the responsibility of the data base or transaction monitoring system chosen for the implementation. It appears that security in this approach must necessarily reflect the ad hoc, uneven, and frequently fragmented security mechanisms available from the commercial products upon which the information system is built. This has usually been bern out by experience.

In sum, civilian agencies are faced with the problem of scarce resources against growing workloads and old systems. These systems contain weak security features; however, they continue to be useful and will pose problems in the future. Good security techniques are known by selective groups within each civilian agency. Making these techniques available will greatly enhance the future security of agency information systems. A stronger policy will be needed to treat the in-place weaker systems where retrenching and security enhancements may be required.

VARIATION IN PASSWORD USAGE

Authentication of the user is of primary importance in information systems and is commonly done with passwords or challenge-response devices, such as a DES calculator. As with any authentication token, the doctrine associated with their use is a critical aspect of their security. A reasonable protocol for password generation, distribution, and use is presented in the NCSC "Password Management Guide"[2], and under the C2 doctrine, each user should possess an individual password to allow accountability to the single user level of granularity.

In comparison to the C2 doctrine, we have found that within a single shared civilian computer system, there is a wide range of password practices. Each information system may command its own doctrine of use; thus, it is not uncommon to find a very sensitive information system, perhaps written using IBM's Customer Information Control System (CICS) to require rigid conformance to the C2 doctrine, while along side it and in the same mainframe is a non-sensitive application whose passwords conform to C1 (group passwords) or less (no password).

Examination of several unclassified systems reveals a number of operational doctrines different from the C2 doctrine. These are listed in Table 1 and are ranked from "least" sensitive to very sensitive. Each of the systems in themselves do not violate given norms of security. The major problem is when they are mixed on a central mainframe using commercial non-secure operating systems.

The first examined in Table 1 is that of a public information and retrieval system which uses no passwords and is available to the general public. This service is somewhat new and is aimed at providing the public with the latest information regarding announcements, rulings, and orders. Originally, terminals to these systems are placed in the lobby of the agency's office and are open to all. However, dial-in mode of access is now beginning to make its appearance. Some experiments have been done with public-to-agency electronic mail in which the mail package runs on the agency's host. The public information systems are commonly implemented using one of the commercial transaction monitors and, hence, the "isolation" one can expect is an open question.

Public information terminals, electronic mail, and public information data bases are attractive in terms of enhanced efficiencies and a tool useful to the public. This service will probably continue to gain popularity in the future and, hence, represent a potential problem area for the security of shared agency hosts.

The second example in Table 1 is the "strong room" mode of password usage. In this example, only cleared individuals have unescorted access to this strong room. The premise is that only persons with the proper need-to-know can enter this strong room; therefore, there is no need for further authentication. This corresponds roughly with the C1 doctrine of group passwords in a cooperative environment. Yet, prudent handling of this unclassified, yet sensitive, information would seem to require accountability at the individual level of granularity. The style of operation in these strong

TABLE 1

PASSWORD USAGE

	Logon Passwords	Information System File
Public Information Terminal Resident in Agency's Lobby	No	No
Public Information and Agency Announcements Available via Public Dial-In Access	No	No
Funds Disbursement System	Yes	Yes
Regulatory Investigative Data Base	Yes	Yes
Confidential Informant Identities [terminal located in controlled area (strong room model)]	Group	Group
Counter Style Service (office model)	Group	Physical Lock

rooms are dependent on the responsible managers and varies from group to group. Tightly knit groups tend to operate as a "skunk works" typically relaxing some rules in favor of the mission. Large groups require more accurate tracking of information and, unfortunately, may not employ adequate methods. It is probably this latter group that will gain from the C2 doctrine.

The third example in Table 1 is the "office" model of password usage. In this model, the public is being serviced by agency personnel from "windows" which contain a terminal hooked into the agency's information system. The agency person may, from time-to-time, leave the terminal. The terminal could then be manipulated by an unauthorized person.

To prevent this type of compromise, some information system designers have introduced the use of physical locks on the terminal which lock the keyboard and blank the screen. Some simply lock the terminal keyboard but do not blank the screen. The agency user removes the key and takes it with him to prevent compromise or alteration.

This third example is important because it brings out an important, but overlooked, authentication problem--transaction-oriented authentication. Common use of account identity and passwords is designed to be session-oriented commonly authenticating an entire session from logon time to logoff or session termination time. Because a session typically exists over a significant period of time, it is acceptable for the user to perform the manual operation of entering the passwords and account name since this is done once per session. The overhead time associated with performing a logon is acceptable because it is done once for the entire session.

Comparatively, a transaction can be viewed as a short unit of work which is performed many times over the period of a session. The operational premise of session logon is that the user will keep control of the terminal for the duration of the session. Yet, there are circumstances where this is not true, especially in the case of the "strong" room model and the "office" model.

What is needed is the ability to authenticate a user based not on a session, but on a transaction basis. Conventional account name/password pairs are inadequate for this role because the logon itself is an example of a transaction.

Some approaches have been suggested for the transaction authentication problem. Chiefly, they center on eliminating the delay associated with manual account name and password entry by using a badge reader to gain this information. The information read from the badge could be appended to an invisible field prefacing the user text record. An unsolved aspect of this issue is the additional processing time added for each account name/password verification. This might best be dealt with by a fast verification technique different from that developed for session authentication.

The last example of password usage shown in Table 1 is the shorter than required password. This is essentially a human factors problem, and it is common to find installations where eight-character random password combinations are systematically changed and result in users writing them down. Some relief can be expected from passwords composed of pronounceable syllables or pass-phrases; however, the doctrine of random string passwords systematically changed is a problem.

To overcome this, some information systems have adopted four-character passwords, while others simply do not change them allowing users to memorize them through long-term use.

It is clear that while passwords are currently a useful authentication technique, the management and human factors overhead associated with them is great. New methods commonly employing DES calculators, badge readers, and smart cards are beginning to make their way into use within the civilian sector but are still relatively expensive. Primarily, they are first considered for very sensitive security applications because they commonly afford two-step (what the user knows and what the user has) authentication as opposed to one (a password).

Each of the examples discussed (the public access system, electronic mail, the strong-room, and office model) illustrates an important principle. This principle rests on the view that user authentication should be tailored to the environment in which it is used. Shorter passwords may suffice if a proper password checking mechanism was also in place. This mechanism would prevent trials by an unauthorized user and would quickly lock-out too many attempts. Including a mandatory delay time also lowers the chances of a successful brute force attack.

In sum, it may not be necessary for every application of passwords to strictly conform to the C2 doctrine. The mode of operation and degree of induced risk should be analysed on a case-by-case basis.

SENSITIVITY AND PROTECTION

Allocating sufficient protective resources is an essential management decision in the civilian sector which must be carefully measured. There are two philosophical views commonly employed for allocation; the civilian agency view and the military or national security view. While utilizing similar methods of risk analysis to allocate resources, these views differ primarily in their understanding of the world. In turn, this is seen as a natural consequence of the mission and environment in which each operates. This section contains a brief examination of some of the issues affecting the estimation of threats and assignment of protections for each.

Devising a protection scheme is a two-step approach. The first step in devising a protection scheme is to have a way to optimally utilize limited countermeasure resources. Countermeasure in this case consists of measures used to enforce system security above what is available with a standard commercial software package or which is available without cost from the information systems data base or transaction monitor. This includes investment in an access control package, enhancements to the operating system, or other security modifications.

The method used for protection estimation is similar to the common average loss expectancy (ALE) approach commonly used in risk analysis/risk reduction. The ALE method is quite popular in the civilian sector for allocating risk reduction funds. It is recognized by most federal agencies as a valid and adequate approach to identifying and ordering the major risks and estimating the sufficiency of a risk reduction strategy.

The ALE is a four-step approach in which the individual asset value and loss event frequency are compared against strategies of backup allocation. In the first step, each asset is identified and a dollar value indicating its replacement cost is assigned. Second, all threats are identified and a probability of a compromise or security-significant event is computed. Third, the probability of an event is combined with the loss value of that asset acted upon to yield a total expected loss value. If this value is computed for critical assets, for example in a computer center, then a manager can decide where best to optimally place scarce contingency resources.

A contingency resource might replace an asset lost in an attack or disaster or it might provide a reduced capability. The cost of replacement or of an interim measure to offset loss of a critical asset is subtracted from the expected loss. Thus, the fourth step is to test different allocation strategies of replacement resources to minimize the expected value of loss.

The ALE approach can be applied to information systems by first calibrating the frequency of a security event against that data. An event of this type could be a compromise, manipulation, theft (removal), or denial of service attack. In turn, the probability of attack is multiplied by the damage done to the asset estimated in dollars. This yields an expected loss value which can be used to allocate funds to a protection strategy.

After a protection strategy has been picked and the ALE recomputed, there is commonly a residual amount of risk which cannot be serviced under a given budget. This risk commonly centers on infrequent or unusual events which are considered improbable; however, frequently it is much more likely that risks can only be partially addressed under a current budget. These "unfunded" risks are present in the systems operation and are commonly judged by management to be acceptable levels of risk for a given operation. These residual risks are treated under the heading of risk management.

While the ALE method has found wide acceptance as a risk objections reduction allocation method, there are numerous objections to it both practical and philosophical. Briefly, the practical objections center on the process of estimating the dollar value of important information in a data base and the frequency of certain types of attacks on that information. An example is an investigative data base containing very sensitive information.

How does one go about determining the value in dollars of such a data base? The replacement cost can be calculated by costing the history of investigative actions which supplied the information. Sometimes it is possible to do this and sometimes impossible as in the case of a data base built up over years; however, the losses go beyond simple replacement. Commonly, compromise of investigative information can result in thwarting an agency mission by divulging the confidential sources and methods used by the government to develop the case. In addition, there may be political and legislative repercussions which often interact in unexpected ways.

Estimating the frequency of an action is also difficult chiefly because there are few solid histories available. Certain sources are available; for example, the Justice Department's Bureau of Justice Statistics, the FBI, and some national trade organizations can give information on occurrences of blue and white collar crime. Yet, aggregating numerous statistical sources which only partially address the specific circumstances at hand necessarily lessens predictive ability.

Both estimating cost and determining threat event occurrence frequencies are processes which cannot be carried out in isolation. The process is usually the culmination of numerous discussions with principals; analysis of documents, budgets and plans; and a considerable amount of estimation. The tentative figures developed by the analyst must be defended to management who may apply the "reasonable man" argument to them. This argument suggests how a reasonable man or disinterested third-party might view the realism of the analyst's numbers. Often, the reasonable man argument would more honestly be described as the organizational man argument.

Philosophically, the objection to the ALE method centers upon the meaning of expected loss. The mathematical notion of expected value allows one to calculate an average value given both the probability of an event and the value of the asset. Yet, the number of samples available to calculate the probability is often limited and carry with a large degree of variance as to often make the calculations meaningless.

A second objection concerns what information an average value has to offer for a specific loss situation. It is based on the average behavior of an event drawn from a large population of events. Average value theory allows an insurance company to calculate what the expected loss is over a very large number of insurers because each of the insurers will be different, some claims will be less and some more. On the whole, the insurers can be expected to act in a more or less randomised fashion within a given set of criteria. Knowing the distribution function for the insurers allows the insurance company to calculate the mean and, hence, calculate its expected loss and adjust its premiums accordingly.

Over the entire sample of computer centers there may be a given sample distribution which indicates government-wide what the distribution of loss is. Each individual case can be expected to be different; some of more and lesser degree. Like the insurance company of the above example it will be possible to calculate an average and determine average loss for that population but what does it say about the individual case? In sum, ALE may be likened to a gambling strategy in which management bets that an event "is on the average" no worse than expected.

A final difficulty experienced using the ALE approach is that it produces a different level of protection for each information asset to which it is applied. This often leads to a fragmented approach to protection and does not produce a general set of protection mechanisms available for future information systems placed upon the host.

The second step in an information security program for the unclassified civilian agencies is to develop a small number of uniform classes for asset value and protection.

Military agencies arrive at a uniform definition of value--the military definition of security relies on damage to the national security as its basis. The damage assessment defines a number of levels named confidential, secret, top secret, and various need-to-know groupings which form the familiar military classification hierarchy.

There is also a uniform degree of physical protection called out in the various military security doctrines. Thus, the elements of the ALE, asset value, frequency of event, type of event, and protection allocation, are standardized for the entire "classified" community. This framework is in place and what is left is compliance determinations.

The military doctrine is set up to counterbalance both individual and sophisticated threats and is skewed to a worst case analysis of the threat. Taking this worst case approach simplifies the security program because it removes the requirement of staying one step ahead of any threat which may evolve on short notice. There are great operation advantages to this simplified approach if funds are available.

In comparison, the civilian sector commonly requires the protection system to expend only enough to counter the average threat facing it. The threat environment facing civilian sector agencies is much more stable and unchanging then the one facing national security establishments; hence, as in life insurance, the gamble is that the agency's information protection approach can bound security risk problems on the average.

A pressing disadvantage of this approach is that, to date, no uniformed classification system has been advanced for the unclassified civilian sector which could allow the streamlining of protection programs. If a uniformed level of protection and threat similar to the military classification hierarchy were available and suitably directed toward a civilian agency's threats and if it received the right recognition from the Executive Branch, information security might be advanced across the entire federal civilian sector.

LEVEL OF PROTECTIONS

Establishing levels of protection entails creating a map between value of a asset, the degree of trust for personnel, and the resulting amount of trustworthiness needed from the software. The mode of operation is important as well, and for purposes of this paper, the multi-sensitivity shared mode is assumed. Other modes such as dedicated, or system high mode, modify the balance and threat environment and, thereby, require a somewhat modified analysis.

The first step in determining the level of protection is to create an asset measurement value system. As discussed before, two approaches are possible--the military classification hierarchy and the civilian ALE method. For purposes of this paper, we have adopted a system which conforms roughly to the military hierarchy but without the national security[2] implications which is presented below.

Military Classification	Proposed Civilian Marking
Unclassified	Non-Sensitive
Military Sensitive	Minimally Sensitive
Confidential	Sensitive
Secret	Sensitive
Top Secret	Very Sensitive
TS/Categories	Extremely Sensitive

This shows a four-level value asset hierarchy with approximate correspondence to the military classifications. It is important to note that this chart shows asset worth by the damage a compromise could do rather than assigning a dollar value to the asset as the ALE would suggest.

The difference between military sensitive and minimally sensitive is that the term "sensitive" is currently used by military and national security organizations to denote unclassified information which impacts aspects of the national security, but, hereto, has been considered unclassified. Information in this category includes national and international financial trends and projections, movement and supply of strategic material, force readiness estimations, positions for international treaty negotiations, and similar information. The civilian treatment of the term sensitive is more in line with the overall sensitivity of the information in a domestic context. Items covered under the civilian usage of the term would be information specifically protected by law, such as grand jury testimony; identities of confidential informants; tax return data; and agency sensitive information, such as confidential correspondence, financial decision-making or disbursement data, and employee salary or medical histories.

The highest category in this chart is extremely sensitive and it has been equated with the military usage of top secret with categories. It is felt that the sensitivity of this information greatly exceeds that which would be considered very sensitive. No commercial operating system, enhanced or otherwise, not specifically designed for the A class should be considered adequate for this type of information and, hence, it must be compartmented and run on a dedicated system. It is not economical to run such information in a shared mode, and the risk from other users, which is difficult to estimate in most normal cases, would be unacceptable in this case.

The ratings displayed in the sensitivity marking table are designed to be used on all civilian agency information, especially computer systems. Having four levels whose value is recognized across all civilian agencies appears to be helpful to the planning and budgeting process as well because it now clearly states the value of the data.

The next step in establishing a level of protection plan is to establish a system for user trust and access. The Office of Personnel Management has advanced a system of clearances for the civilian agencies which is similar in scope to the military clearance structure which is shown below:

Military Information Classification	Proposed Civilian Information Classification	OPM Personnel Clearances
Uncleared	Non-Sensitive	Non-Sensitive (NS)
Military Sensitive	Minimally Sensitive	Non-Critical Sensitive (NCS)
Confidential	Sensitive	Non-Critical Sensitive (NCS)
Secret	Sensitive	Non-Critical Sensitive (NCS)
Top Secret	Very Sensitive	Special Sensitive (SS)
Top Secret Categories	Extremely Sensitive	Critical Sensitive (CS)

The mapping of the OPM clearance structure to civilian information classifications and the military classification hierarchy is simply an initial attempt at establishing such a system. Many civilian agencies currently require the OPM clearances for their computer staff, commonly the higher clearances, because of the wide types of information which they handle. Users, such as data entry contractors or clerical personnel, usually require either the non-critical sensitive or special sensitive, depending upon the sensitivity of the information they are handling. Extremely sensitive information and its OPM clearance--Critical Sensitive--are used for specialized personnel handling the most sensitive data.

The last step in creating the mapping of value, access, and protection is to create a mapping between value, exposure, and software protection.

A suggested ranking for the unclassified sensitive world is shown below.

Civilian Information Classification	TCB Class	Mode of Operation
Extremely Sensitive	C1	Dedicated
Very Sensitive	C2	System High
	C2-ae	Shared-multilevel
Sensitive	C2	System High
	C2-a	Shared-multilevel
Minimally Sensitive	C2	

There are four entries in this table which indicate the four levels suggested previously. Column 3 indicates the mode of operation. There are three modes indicated here: dedicated, system high, and shared-multilevel. First, dedicated mode describes the case where the available hardware and software are not secure enough to guarantee good security. This mode is reserved for the most sensitive missions.

While we have indicated a C1 "group" style environment being adequate, prudent administration should call for anti-fraud and anti-white collar crime mechanisms to monitor and control the use of information by cleared employees.

There are two versions of C2 mentioned in this chart: C2-a ("augmented") and C2-ae ("augmented" and "enhanced"). The chief difference between these two is that C2-ae suggests the use of privacy encryption (DES) on all very sensitive information.

Additionally, encryption would be applied to files and used to provide a higher degree of access control and authentication above those commonly required by C2. Encryption of data without a firm base of trusted software surrounding it limits its ability to withstand attack; however, it is a tool which should not be ignored, especially in that multilevel trusted systems may not be available in the near future, especially for the civilian sector.

TCB CLASS C2 AND ENHANCEMENTS

The NCSC guidelines specify a C2 class system as the minimum protection strategy for unclassified information which requires need-to-know separation. Additionally, the NTISSC staff has set C2 as a target for federal agencies. This is a necessary but difficult task for many federal agencies, but it must be pointed out that C2 may be inappropriate for certain sharing situations. Primarily, these situations involve multi-sensitive sharing between information systems which hold very sensitive information and those which information systems used to store minimally sensitive information on the same mainframe.

A very high reverse correlation in civilian agencies between the amount of information to process and its sensitivity is almost an exponential relationship. As described previously, the lower the sensitivity, the lower the clearance levels and looser the security administration in general.

There are three reasons for looser security controls in these lower sensitivity systems. First, authentication and access control restrictions are relaxed in favor of getting the job done. In the second case, administering very large numbers of users who report to different chains of command and are distributed over large distances is very difficult. These leave numerous opportunities for abuse of passwords and access privileges.

Third, there is a tendency for users to accumulate file access privileges awarded for files or file categories in order to meet a particular need, which are, however, not surrendered and simply accumulated. Thus, it is not uncommon to find users with access rights to large portions of a system without a current need-to-have for these accesses.

While these problems can be solved with good security management practices, it must be recognized that decentralized and, hence, fragmented security administrations do exist. It is also unreasonable to suggest that the situation will change dramatically in the near timeframe. Sharing computer resources among a large population of users will bring with it a higher risk, precisely the situation addressed by the NCSC guidelines for classified users.

What is proposed is the development of a new middle class between C2 and B1 which would contain many of the stronger features of B1 but would continue to rely on non-mandatory access control structures in favor of the lower cost rule-based access control packages. The proposed class is labeled C2-a, which for discussion purposes would serve as an interim step for the civilian agencies until stronger multi-level systems were available from the Evaluated Products List (EPL) at a reasonable cost. While not fully developed, the major requirements of C2-a are outlined below.

There is a fundamental breakpoint in the TCB between the C2 and B class of systems which is seen in all the security requirement areas. This demarcation is reflected in a number of ways, the most obvious of which is that Class B1 is the first class where mandatory controls, labels, and a security policy are required. Comparatively, Class C need provide only discretionary controls.

A second important difference is the ability of the system to withstand penetration. Class C2 requires that obvious flaws be identified and removed while Class B1 requires detailed study of the operating system code in addition to the required live testing. Elimination of obvious flaws required by C2 leaves numerous more subtle flaws untreated, yet B1 requires these to be removed. A skilled attacker could find a C2 system susceptible to penetration by flaws which might be well known in the system programming community. Class B1 requires these flaws to be removed.

Above those required for Class C2, improvements required by C2-a are improved audit trails, better access policy, markings on multi-sensitivity computer output devices, and a private address space for the system security mechanism. Each will be discussed below.

First, improved audit trails are critical to good security since they frequently provide the only record of what actions occurred during a security breach. They have proved decisive in locating and, in some cases, prosecuting an offender and should be carefully designed on a new system. There are two modes of analysis of an audit trail: post-mortem and defensive analysis.

Post-mortem analysis takes place when a security breach has occurred and a time history is being assembled of the event. Participating in such a post-mortem reconstruction is often the best teacher of what information to include in an audit trail and how it should be organized. Since the audit trail may be introduced as part of a court proceeding, its designers should also have a knowledge of the rules of evidence.

Defensive analysis occurs by systematic analysis of the audit information on a routine basis. This allows a security officer to identify suspicious activity as it happens. Frequently, too much irrelevant data is available preventing any serious analysis of the audit information. Some researchers have suggested employing artificial intelligence techniques to automatically analyze the audit information for problems.

Beside acting as one of the reliable records of past events, the knowledge of the existence of a well-designed audit trail deters white collar criminals[4] by simple surveillance of the system. This increases the certainty of being caught and successfully prosecuted. In sum, an enhanced and well-designed audit trail, together with analysis tools, is certainly necessary in any multi-level sharing endeavor.

The second requirement of C2-a is improved security policy. It is first necessary to compile and have approved an agency-wide security policy. In this paper we have advanced using the four levels of sensitivity as a starting point. Federal law, executive orders, special department-wide instructions, and agency orders often form other access restrictions. In addition, many agencies also partition access by program and project.

Access control packages are beginning to include provisions for controlling data base and transaction monitor file access, yet many projects are reluctant to surrender their privileges to a central authority. A way around this dilemma is to use local project-oriented access rule managers in addition to the computer center security administrator. Each access rule manager would have responsibility to his own project.

A useful extension, which currently does not exist, would be a rule protocol which allowed each party to levy constraints on what kind of accesses can be granted. Thus, the computer center security officer could restrict the local access rule managers from adding new accounts to the system, while the local access rule manager could restrict the security administrator from granting access to files under his jurisdiction. Split granting authority, special authorization digital signatures, and other devices including anti-fraud aides might be included to create a rule-based, power-sharing structure to meet the needs of any two parties, cooperative or distrustful.

The third item necessary for enhanced sharing mode is sensitivity markings on multi-sensitivity I/O devices and the labeling of input and output hard-copy. This requirement is chiefly to avoid mishandling of hard-copy input and output media and to ensure proper control of terminals. Many large computer centers commonly have large printer bays in which a number of high-speed printers are used. Printouts are commonly routed to a printer based on the type of paper in the printer, for example, multi-part "carbon" paper. It is quite typical to find computer centers routing jobs of various sensitivities to a printer depending upon the paper forms needed. Thus, very sensitive printouts are handled as non-sensitive until received by the I/O control clerk. Additionally, items like memory dumps are typically not controlled at the level of the data they contain.

Formal object labels, user clearance data bases, and reference monitors are the heart of a mandatory access control structure. To have a reference monitor at all seems to require object labeling and, by implication, a user clearance data base to allow the reference monitor to apply the security model. It is a matter of disagreement at this point whether an access control package can suffice for enhanced sharing or if indeed the full suit of labels, reference monitors, and clearance data bases are required.

If other requirements can be satisfied, then it seems adequate to settle upon an enhanced rule-based access control system to enforce sharing. The strength of the mandatory access is its ability to mediate all accesses at a basic level. At the B1 level the required assurances are not yet developed to the point where multi-level sharing can be trusted as is the case in a B2 environment. Therefore, the essential aspect is the reference monitor's ability to mediate all accesses. Within the C2-a environment, an enhanced rule-based access control package might be adequate.

Lastly, a requirement for multi-level sharing is that the security mechanisms not be subvertable by a malicious user. One of the best ways to do this is to place the access control mechanism in its own address space and take measures to protect critical information that it uses. Some access control packages share certain reserved system address space virtual memory along with other special routines and the operating system. Information modification, either by a trojan horse routine included within the operating system or commercial package or an inadvertant modification due to an error, could disrupt the access control mechanism. Further research would be needed to determine the extent of this vulnerability in the open system environment common in civilian agency's data centers.

This section has outlined the argument for an enhanced level of protection beyond the C2 level but without some of the structures required for the B class. The C2-a class was advanced to support cases where very sensitive information is handled within the same commercial mainframe and operating system in the presence of numerous minimally cleared or controlled users.

C2-a would contain stronger resistance to penetration than C2 by elimination of subtle flaws. C2-a operating systems would have enhancements of their audit trail capabilities, access control down to the information system-owned file level, and output marking capabilities. In sum, each of these measures reduces some aspect of risk associated with multi-sensitivity sharing and is proposed as an interim step which could be accomplished by the civilian agencies sector until more products become available on the EPL. The discussion of C2-a presented here is simply an overview of the requirements the proposed class would need. Additional research would be needed to clearly define all aspects of this proposed class.

CRYPTOGRAPHIC FILE PROTECTION

An often overlook technique of protection for multi-sensitivity sharing is file encryption[5]. File encryption is not currently considered useful in satisfying the requirements of the trusted computer base and has been partially neglected in favor of a trusted software approach. File encryption is still useful for providing an extra layer of safeguard in a computer system and is suggested as an additional security tool for multi-sensitivity sharing situations.

File encryption is attractive because it is one of the only means available to prevent even a skilled system programmer from browsing stored information. It is, therefore, attractive to organizations with the most sensitive data. File encryption prevents browsing by user organization members with general authorization to view all files within their organization except specific ones.

There are four major points to consider in selecting a file encryption technique for the C2 level:

- Cryptographic encoding method
- Target file organization
- Key management
- Trustworthiness of the encryption routine

The first element of file encryption is to consider the encoding method. Cryptographic encoding is commonly applied in three ways:

- Block or electronic code book mode in which a group of bits which form a block are enciphered together and the enciphered output is a block of similar size.

- Chaining methods in which samples of plain text and/or ciphertext from previously enciphered blocks are mixed with current information during the encipherment process. Block chaining prevents certain types of spoofing attacks but also propagates errors across some amount of data.

- Additive method in which a cipher key stream is generated independently and combined with the file information on a bit-by-bit basis. The additive method has the advantage of not propagating errors but suffers from a need for precise synchronization.

Selection of algorithm for a file encryption capability is limited to NSA-approved algorithms for government use. Perhaps the best known example is the data encryption standard algorithm (DES) which is finding popular use in privacy applications. The drawback in using DES is that it is computationally expensive for software implementations, and this must be factored into response time calculations for interactive applications.

NSA has advanced several new algorithms under the Commercial Cryptographic Endorsement Program (CCEP), which are targeted primarily at the communications marketplace. The Type II algorithms are meant specifically for the unclassified area; they are available only as integrated circuits to approved manufacturers and are not available in software form.

Thus, while DES will be gradually withdrawn from the federal marketplace in favor of the CCEP Type II algorithms, no substitute has been suggested for software versions of DES. Some authorities have suggested that a way to approach the file encryption problem is to develop a fast hardware box containing the CCEP Type II algorithm. The "crypto-box" would be used as a controlled peripheral device to a computer system and would provide a faster replacement for DES. To my knowledge, no work has been done to develop such a box or standards for its use.

File organization is important to a cryptographic file security tool. Computer files have different record organizations such as sequential, random access, relative record, indexed sequential, and proprietary methods developed by data base manufacturers. Fitting an encryption technique to each specific type of file organization is necessary in that ciphers commonly must be started at a mutually agreed starting point which is difficult for short and varying length records. This is in some respects similar to the problem of end-to-end encryption in an X.25 network where each packet must contain its own initial fill value.

Therefore, in providing a file encryption package, each file must be considered separately. In archive cases where an entire magnetic tape is to be enciphered, chaining schemas work well because of the long stretches of data while short records, which are randomly organised, require completely different enciphering methods. The requirement for flexibility precludes a rigid single format and some adaptable techniques are required, perhaps as a set of subroutine calls to a "trusted" encryption facility.

Key management is the third aspect of a file encryption technique. This is perhaps one of the most difficult areas to surmount because of its crucial nature. The nature of the current modern information systems is to provide access online to numerous individuals sometimes distributed over a wide geographical area. Keys must be distributed to each of these parties, and the number of issued keys would grow quite large. Indeed, what we are faced with here is a problem not unlike that of a secure packet switching system in which keys grow exponentially with the number of network nodes.

The solution advanced with the packet-switching networks and which may have application for a file encryption tool is a key distribution facility (KDF). The role of the KDF would be to enforce an access protocol on each user, perhaps employing a key shared between the KDF and the user. The actual file encryption key would not be shared with the user but might be derived from the user's key, a file key, and perhaps a local security key.

Design of such a system and indeed its protocol will need development while implementation could employ either conventional key management techniques or perhaps public key ideas. It appears that the KDF idea will probably prove a way to solve this difficult problem.

Lastly, assurance is the fourth issue in design of a strong file encryption technique. We are faced with a problem similar to developing trusted software for the TCB in that the techniques are similar. It is for this reason that the C2-a system proposes stronger penetration testing than simply elimination of obvious flaws found in C2 requirements. Yet, there is a delicate tradeoff required in arriving at a balanced assurance level for the file encryption tool.

File encryption can provide good privacy and authentication methods when there is high risk and when other techniques are unavailable. File encryption is not being discussed here as a replacement for trusted software chiefly because it fills a somewhat different function and serves as a useful adjunct. In sum, the techniques and products for file encryption on unclassified sensitive computer systems are not as well developed as one would wish; however, it is one of the proven tools which are available where other forms of risk reduction are not available.

CONCLUSION

This paper has presented a discussion of security factors affecting unclassified sensitive civilian agency systems.

The multi-sensitivity sharing problem was reviewed and two essential questions were proposed: first, can the information system adequately control the users authorised to use it, and second, can the operating system prevent users of one information system from accessing files and resources of some other system? It was shown that both these issues cannot be given an unqualified answer lacking multi-level trusted software; yet, it is possible to substantially reduce the risk of multi-sensitivity sharing by good software security engineering and basic security enhancements to the operating system.

In this respect the notion of an enhanced version of the basic C2 requirements was advanced with the goal of introducing improvements to better manage a multi-sensitivity job stream and improve the system's resistance to penetration. Additionally, file encryption was suggested as a further way to limit risk in systems where there is a large difference between the lowest clearances and the highest classification of data.

Primarily, the C2-a suggestion is viewed as an interim step which could provide better security in the period until true multi-level products are available from the EPL at a justifiable cost. This may be a considerable period of time because researchers and manufacturers are targeting the classified market first.

Lastly, it is important to note the difference in culture between military and civilian agencies. Each has evolved in a culture facing significantly different problems and, hence, responses and perceptions are different. If the body of military security knowledge is to be of value to civilian agencies, it must begin by reformulating its associated doctrine of use. Security programs which do not conform to an organization's culture will ultimately be expensive to administer and vulnerable.

REFERENCES

References Cited

[1]DOD Computer Security Center, Password Management Guide, CSC-STD-002-85, April 1985.

[2]Executive Order 12356, National Security Information, April 1982

[3]Office of Personnel Management, Federal Personnel Manual, Position Sensitivity, Basic Installment 311, January 1984.

[4]Department of Justice, Bureau of Justice Statistics, Computer Crime, Electronic Funds Transfer Systems and Crime, U.S. Government Printing Office, Washington, D.C., July 1982.

[5]IBM Corporation, Data Security Through Cryptography, GC22-9062-0, October 1977.

Other References

DOD Computer Security Center. DOD Trusted Computer System Evaluation Criteria, CSC-STD-001-83.

DOD Computer Security Center, Guidance for Applying the DOD Trusted Computer System Evaluation Criteria in Specific Environments, CSC-STD-004, June 1985.

DOD Computer Security Center. Technical Rationale Behind CSC-STD-003-85, CSC-STD-004, June 1985.

McPhee, Operating System Integrity in OS/VS2, IBM System Journal No. 3, 1974.

TOWARDS A DISCIPLINE FOR DEVELOPING VERIFIED SOFTWARE

William M. Farmer
Dale M. Johnson
F. Javier Thayer

The MITRE Corporation
Bedford, Massachusetts

Abstract

In this paper the formal verification of computer systems and software is viewed as an endeavor in applied mathematics. It is argued that a formal verification should consist of three separate but interacting processes: a modelling process, a theorem proving process, and a review and acceptance process. Suggestions are made for improving the development of these processes. Taken together, they outline a proposed discipline for the development of verified software. The ideas presented were principally, though not exclusively, motivated by the authors' work in reviewing the design verification of the Restricted Access Processor (RAP). Examples are drawn from the RAP verification to support our suggestions for improving formal verification.

1. INTRODUCTION

The main purpose of this paper is to propose a discipline for the development of verified software. Our comments in this paper are motivated in part by our recent experience reviewing the design verification of the Restricted Access Processor (RAP) (cf. [3], [7]). We also draw some examples from the RAP verification to support our views. The paper attempts to describe verification as an endeavor in applied mathematics. Though this viewpoint is not completely new (cf. [1], [10]) and might even be regarded as the obvious one to take, from our review experience we are led to believe that the exact consequences of taking this view are not fully and clearly understood. In particular, perceiving verification as applied mathematics requires a clear differentiation between the following two processes:

(1) *Establishing formal mathematical models of natural-language requirements or specifications.* In the case of design verification for secure systems, these models are usually called the formal security model and the (formal) top level specification (TLS). We refer to this as the modelling process, which in our view is perhaps the most critical part of the verification, yet it is apparently the least understood.

(2) *Using mathematical techniques to reason about the formal models obtained by the modelling process.* In the design verification of the RAP, this reduced to proving formally that the TLS satisfied the formal security model. We call this the theorem proving process.

We realised in our review of the RAP that the distinction between the modelling process and the theorem proving process is especially important from the reviewer's perspective, since the tasks involved in each of these processes are to be understood and judged in very different ways. Yet while these processes are distinct, it is inevitable (and definitely beneficial for the verification) that they will interact with one another.

In addition to these two processes we believe that it is useful, in analogy with similar validation processes occurring in the mathematical sciences, to include a third interacting process as well:

(3) *Reviewing and accepting the verification.* Generally, this means ascertaining that the verification satisfies requirements agreed upon by the customer and the verifier. Requirements may include, for example, the use of automated tools. Another relevant and more important requirement is soundness of the logical principles used in the verificatior In our view, part of the review process should allow for interaction between the reviewers and the verifiers.

We gratefully acknowledge the support of the Rome Air Development Center under contract number F19628-86-C-0001 during the preparation of this paper. We would also like to thank E. Bensley, J. Millen, P. Tasker, and J. Williams, who offered guidance in the formulation of the ideas presented in this paper.

2. THE MODELLING PROCESS

Mathematical modelling is a crucial process in the task of formal verification. For example, in the verification of secure systems, a formal security model for the security policy is constructed or, in some cases, provided (e.g., the Bell-LaPadula security model). In this section we discuss various aspects of the modelling needed in verification, using our findings from reviewing the design verification of the RAP to develop examples and special points.

In general, models provide a description of some real-world phenomenon. By "phenomenon" we mean something very general. A phenomenon may be a process or a system; even a natural language description of a system or process is a phenomenon. A fundamental aim for constructing a model is to allow the use of formal deductive techniques on the model to gain some new information or conclude something about the phenomenon. This aim requires that models be comprehensive in the sense that they contain all information necessary for applying these formal techniques. It should be emphasised that formal deduction is clearly distinguished from other forms of evidence, such as empirical evidence, so that the requirement of comprehensiveness is quite important.

Another highly significant aim of the modelling process is to make the phenomenon intelligible to others. In order to achieve this, the models have to be clear and thoroughly explained. Models that resemble computer code do not meet these goals.

The process of building useful models is one of the most difficult in all of science. The model-builder has first to select carefully the tools and techniques from mathematics that seem the most appropriate for presenting the model. Most critically, he must decide how to represent elements of the phenomenon with mathematical constructs.

The model should be a clear portrayal of the phenomenon, so that it can be accepted. Acceptance of a model is based on the collective experience of the researchers doing modelling and also on subjective factors, such as mathematical taste. A precept that is universally true is that models are meant to be understood. Questions of style and format are not to be brushed aside as technically irrelevant. Moreover, specific sciences have developed special methodologies for validating models. These methodologies generally rely on experimentation and statistical sampling; even some form of disciplined introspection may be used.

Unfortunately, no modelling methodology has, to our knowledge, been successfully developed for the young science of verification. The lack of a methodology makes modelling even more difficult.

We must emphasise that by the term "modelling" we do not refer exclusively to the construction of the formal security model, though this construction is a significant part of the modelling process in some verifications such as the RAP. We must also include the writing of the formal top level specification (TLS) as a part of the modelling of the system.

The modelling required for the design verification of the RAP is typical of that needed in verification. We can identify the parts of the modelling process in general as follows:

(1) Selection of a methodology for the verification, such as the Hierarchical Development Methodology (HDM) [6]. This selection has significant implications for the verification. HDM was used for the RAP verification. (Other possible methodologies are Gypsy and Formal Development Methodology. Also, an Enhanced HDM has recently been released.)

(2) Construction of a formal security model derived from a security policy. The purpose of this model is to formalize natural language requirements concerning security. As part of the modelling process, the functioning and adequacy of the model should be explained. In the case of the RAP the formal security model was derived from an Air Force security policy. Some explanation of the functioning of the RAP accompanied the formal model.

(3) Characterization of the design by writing a formal top level specification. The TLS was a large and significant part of the modelling for the RAP verification.

(4) Generation of conjectures during the modelling process, which then need to be investigated during the theorem proving process. In the case of the RAP verification we found that it was necessary to make the exact nature of these conjectures as clear as possible.

(5) Justification of the decisions taken in steps (1)-(4), in order to advance the (implicit or explicit) claim that the modelling is adequate. Unfortunately, it is often the case that this aspect of the modelling is not adequately carried out.

We have prepared some suggestions for improving the modelling process in verification. These were in part prompted by our examination of the modelling done for the RAP verification. In looking at the modelling in the RAP verification we were particularly concerned with the need for adequacy, comprehensiveness, intelligibility, and simplicity. These are highly desirable features that should be considered in the modelling done in verification. Our suggestions are intended to help verifiers make these features a part of their verifications.

(1) Explain and carefully justify fundamental decisions about the modelling.

Throughout a verification project, but more especially near the beginning, the verifiers should attentively think about the modelling needed or being done. Decisions about the modelling should be carefully documented and justified. At the outset of the RAP verification, certain modelling ideas had to be established, i.e., decisions had to be taken about how to portray the actual RAP (the reality in this case) as a mathematical model. The RAP is a processor guarding the data link between the Network Control Center (NCC) and the NASA Communications Message Switching System (MSS). Its purpose is to prevent uncleared users from accessing classified information or facilities available through the NCC. A security policy had been provided by the Air Force and the architecture of the system hardware had been developed. The basic modelling problem was to find mathematical constructs that reflected the chosen architecture of the hardware and the intended security of the system. The verifiers decided to model the operation of the RAP conceptually as sequences of events that passed over a (conceptual) security perimeter. The selection of a particular security perimeter and a particular way to portray the flow of events is a fundamental modelling decision. Verifiers must not only understand the nature of this basic decision about modelling, but be able to justify it as well.

(2) Give broad explanations of the models and, if possible, key information about the process by which they were derived.

Broad explanations of entire models are extremely helpful to a reader or reviewer. Moreover, information about the genesis of the models can illuminate the models themselves. During the construction of a formal model various modelling decisions are made. These are reflected in the final constructed model, but often in obscure ways. The key information about the construction of the models should be preserved in an abbreviated form in the documentation.

In the case of the RAP we found that the documentation, though substantial, could have contained more information about the ideas behind the actual construction of the two main models, the formal security model and the formal top level specification. To take a simple example, we found that one very large definition in the formal security model could be reduced to a pair of tables. Once these tables were constructed, the formal definition became much easier to understand.

(3) Choose a methodology that is adequate to formalise the notions that need to be modelled.

This choice is a very difficult matter. One wants to choose an adequate methodology for a verification, but at present there are only a few from which to choose. A fundamental modelling decision taken for the RAP verification was to adopt the Hierarchical Development Methodology (HDM) [6]. This decision had many implications for the modelling process. Most notably it implied the adoption of the sequential state machine model, a basic part of HDM, in the modelling. For this general model concurrency is not so easily taken into account. Hence, it is at least questionable whether this sequential model is adequate to deal with the reality of the RAP. Arguments ought to be given for the (implicit) claim that the chosen methodology is adequate.

(4) Try to develop a formal model that has a direct and clearly understood relation to the English policy statement or English requirements specification.

The formal security model was a very significant part of the modelling for the RAP [2]. The purpose of this formal model was to capture the Air Force security policy in a succinct and correct way. The model was based on event histories and was written in the specification language SYSPECIAL, a variant of the SPECIAL of HDM. The heart of the model consists of a hierarchy of definitions of predicates on event histories, with a single predicate (MBPS_OK) at the top of the hierarchy representing the desired security invariant.

In order to facilitate the construction of the formal model a shortened form of the Air Force security policy was developed, called the "derived security policy". This was undoubtedly a great help in constructing the formal security model, in particular, in seeing how the Air Force policy should be related to the model. The derived security policy is a terse English-language statement of the security requirements that is closely related to the formal model; in many instances there are direct (one-to-one) correspondences between words of the derived policy and functions or predicates of the model. The model would have been even better if it could have been a simpler formalisation of the policy with more direct correspondence between policy and model. However, formalisation is a very difficult art.

In general, formal models should be made as simple as possible and the relation to English-language requirements specifications should be made as clear as possible through informal explanations in the documentation and perhaps through the construction of derived policy statements. One can see the advantages of a derived tersely-worded security policy statement in the case of the RAP. Generally an English statement or specification should be as simple as possible.

(5) Definitions in a model should have a hierarchical structure and this structure should be presented fully.

The definitions of the formal security model for the RAP were arranged in a hierarchy. This arrangement is certainly a good one. It is one that can be used to good effect in modelling in verification. However, it is useful to have as much information as possible about the hierarchy. The hierarchy effectively indicates a "flow" from the most general to the least general, revealing a great deal about the structure of the model. The verifiers of the RAP might have given more information about their hierarchy. Their diagram of dependencies in the hierarchy was reduced to a brief summary in the documents. A general explanation of a hierarchy of definitions can be very helpful as a supplement to the explanations of the individual definitions found in the hierarchy.

(6) Use nonprocedural forms of expression.

In our attempt to understand the formal security model for the RAP we were led to develop our own intermediate mathematical model and explanation. We found that it was very helpful to remove the recursions from the basic definitions in the formal model and state these with the aid of quantifiers and logic. The formal model as given effectively had a mix of procedural description (the recursions) and strict logical description. This mix was not always conducive to providing a direct and clear exposition of the model.

It was only by constructing our own intermediate mathematical model for the given formal security model that we could begin to see the relation between the English security policy and the given formal model. This intermediate model allowed us eventually to decide that for the most part the policy was correctly reflected in the given formal model.

The modelling done in constructing the TLS for the RAP [8] had some special problems, partly associated with adoption of HDM. We found the TLS at times quite difficult to understand. We have a number of suggestions for improvement in writing these kinds of specifications. In the following we assume an understanding of the terminology of HDM.

(7) Use homogeneous data types, whenever possible.

To avoid confusion, use homogeneous data types. If for some reason the use of homogeneous data types is impossible, pending data types should be considered.

(8) Describe state transitions as simply as possible.

The effects of O-functions on individual V-functions should be easily understood. Ideally, the effects of an O-function should be of the simple form:

$$V = F(W),$$

where V, W are V-functions and F is some simple function. The functionality of O-functions of this sort is manifestly clear to a reader.

(9) Provide an information flow diagram.

The flow of information between V-functions should be clear. Ideally, one should be able to represent the flow induced by an O-function as a directed graph. The nodes of this graph correspond to V-functions and the edges correspond to assignment statements. The graph provides a clear understanding of the general architecture of the TLS.

(10) Give adequate explanations of the relations among the models.

This suggestion brings up the issue of comprehensiveness of the models. One of the goals of the RAP verification was to prove that the TLS satisfied the security requirement or predicate formalised in the formal security model. This formalized security requirement is essentially a predicate on finite sequences of "events". Since sequences of events do not constitute part of the state of the TLS state machine, it is not clear how to interpret the assertion that the TLS satisfies the security predicate.

An interpretation can be made by associating event histories with certain sequences of O-functions or OV-functions. These sequences are the possible execution sequences of the TLS state machine. The assertion that the TLS satisfies the model then means essentially that for every execution sequence, its associated event history satisfies the formal security predicate. However, this correspondence is not a part of either the formal security model or the TLS. How one associates an event history to an execution sequence is a problem of modelling. In the case of the RAP, however, event histories were introduced as part of the theorem proving stage in a manner which seemed to suggest that one could prove that the association chosen was the correct one. Nevertheless, this association was especially problematical, since crucial assumptions about concurrency were implicitly made. In general, the omission of the relation between models means that the modelling process is not as comprehensive as it should be.

3. THE THEOREM PROVING PROCESS

The second process of formal verification is the theorem proving process. In this process mathematical proofs of the conjectures formulated during the modelling process are constructed and analysed. These proofs serve two functions:

(1) To determine whether the conjectures formulated during the modelling process are true.

(2) To clarify the meaning of these conjectures.

The first function is well understood. No doubt it is the part of formal verification that has received the most attention. The second function is often ignored. It is, however, essential for identifying inappropriate or incorrectly formulated conjectures, and thus spotting apparent errors in the modelling process.

Unlike the modelling process, the theorem proving process is a completely mathematical endeavor. Proofs of theorems are developed in a well-defined mathematical theory (created by the modelling process), in which there is no direct mention of the real-world application.

As part of a verification, mathematical proofs can provide a level of assurance for the correctness of a conjecture that is not obtainable by traditional means of software testing. Nevertheless, mathematical proofs are not infallible. Their validity must be ultimately grounded in some kind of critical process. In mathematics, this critical process occurs within the community of research workers.

Because proofs used in formal verification tend to be long and complicated, verifiers usually try to construct them with the aid of machines (theorem provers, proof checkers, simplifiers, etc.). This approach is certainly good and probably necessary. However, without care it can become an obstruction to the theorem proving process, leading to results such as the following:

(1) Theorems are proved without being clearly understood.

(2) Opaque calculation is given instead of careful argument.

(3) Proof analysis is given less emphasis than proof construction.

(4) Conceptual simplification is overlooked.

(5) Errors in the formulation of conjectures are not discovered.

(6) The fallibility of proofs is forgotten.

If verifiers are to construct good proofs with the assistance of machines, they need to have a clear understanding of what a verification proof should be. We feel that there are six basic goals that a verification proof should attempt to achieve:

(1) A verification proof should clearly state the theorem it purports to prove.

To any mathematician this goal is so obvious that it hardly needs stating. Nevertheless, achieving this goal is essential to any good proof. A proof's value is diminished in proportion to the lack of clarity in the statement of the theorem.

(2) A verification proof should increase one's confidence in the truth of the theorem.

This is clearly the major goal of any proof. It is important to remember that proofs can never give an absolute guarantee of correctness.

(3) A verification proof should be rigorous.

The one thing that separates formal verification from traditional ways of testing software and computer systems is that formal verification attempts to show something is correct with a rigorous proof. A rigorous proof strives to use only well-defined concepts and to have no loose ends. Nothing is ignored or left to chance.

(4) A verification proof should clarify the meaning of the theorem.

It is a rare luxury to begin proving a conjecture that is correctly formulated. This is true in general mathematics as well as in formal verification. Thus it is very desirable that the process of proving a conjecture helps to correct the statement of the conjecture itself. The ideal process goes like this: a (partial) proof of the conjecture is constructed, it is analysed, the conjecture is modified, and the process is begun again. The process ends when one is satisfied that a complete proof of an appropriate and correct conjecture has been obtained. (Cf. [5] for further discussion of this "dialectical" process.) In order for this process to be successful, it is necessary that verification proofs elucidate the meaning of the theorems they prove. This cannot be done by proofs consisting merely of a long series of opaque logical calculations.

(5) A verification proof should be maintainable.

Software and computer systems need to be modified virtually on a continuous basis. Hence, verification proofs should be modified whenever the things they verify are modified. In other words, verification proofs should be maintainable just as computer systems should be maintainable.

(6) A verification proof should be machine checkable.

Verification proofs tend to be long and complicated. One cannot expect to check them by hand without making mistakes. It is reasonable to expect that many of these mistakes would not occur when a proof is machine checked. Although it is desirable that a verification be machine checkable, it is not necessary that a verification proof be machine generated. (Of course, it is often useful to construct parts of a verification proof with the aid of a machine.)

The state of the art of verification proofs falls significantly short of these six goals. We believe that this is due in part to an inadequate understanding by verifiers of what proofs should be and what role machines should play in proof construction.

To help illustrate how real verification proofs satisfy (and fail to satisfy) the goals we have stated above, we shall briefly examine the verification proof for the design of the RAP. The RAP verification proof is a good example to consider because, although it certainly is one of the best large-scale verification proofs produced to date, it exhibits some of the deficiencies that commonly plague verification proofs.

The principal theorem of the design verification of the RAP can be stated as follows:

THEOREM. Every implementation of the Top Level Specification (TLS) of the RAP satisfies the requirements formulated in the formal security model.

This theorem is only stated informally in the RAP Verification Results Report [9] and its mathematical meaning is not explicated at all.

The proof of the theorem breaks up into two parts:

PART A. The proof that the theorem holds if the assertions of an Augmented TLS (ATLS) are invariants of the ATLS.

PART B. The proof that the assertions of the ATLS are invariants of the ATLS.

These two parts are handled very differently. Part A of the proof is an informal mathematical argument, which is given very little attention relative to Part B. Moreover, the argument is flawed because part of the modelling process (the construction of the ATLS from the TLS) is mixed up with it.

Part B is of a completely different nature from Part A. It is essentially a series of 36 very detailed formal deductions. The formal deductions are not actually given in the Verification Results Report [9]; instead logs are given of the theorem prover commands used to construct the deductions.

Part B does a reasonably good job of satisfying the goals of rigor (3) and machine checkability (6). Its success results from being a formal proof constructed (and checked) with the use of a machine. Since the logs are modifiable and reusable, part B also contributes to the goal of maintainability (5). The logs, however, are opaque. They do not help one to understand the subtheorems they prove, nor do they communicate the mathematical meaning of the deductions.

The lack of perspicuity in the formal deductions means that one's confidence in the claim that the ATLS assertions are invariants of the ATLS is almost purely a matter of faith in the MUSE system, the theorem proving system used to construct the formal deductions. The MUSE system [4] was developed by Sytek, Inc. It is a competent and, in many ways, admirable theorem proving system. However, although the MUSE system appears to work correctly, it has not been formally verified (like all such systems) and it is relatively untested, having thus far been used on only one large project. As long as the MUSE system itself is not verified, genuine confidence in it can only come after it has been used by several different parties on several different projects.

In summary, the RAP design verification proof is composed of an informal part and a part constructed with the aid of a machine. The first part received only cursory attention. The second part was carried out in great detail, but with too much reliance on automated reasoning tools. Although the RAP verification proof is successful in several ways, it illustrates some common shortcomings of verification proofs:

(1) Insufficient attention is paid to the informal parts of the proof.

(2) Justification for modelling decisions is presented as part of the verification proof.

(3) Formal deductions are not presented in an understandable form.

(4) Too much reliance is placed on unverified theorem proving tools.

We finish our discussion of the theorem proving process by giving five suggestions for constructing good verification proofs.

(1) Use hierarchical construction.

To be readable and understandable, long proofs must be constructed in a hierarchical manner. This is eminently true for verification proofs, which tend to be oppressively long and full of minute details. The components of a hierarchical proof should be subproofs of the form

by the argument A, C follows from $H_1,\ldots,H_n$.

At the top level C is the conjecture that is proved, and at the bottom level the H_i's are the hypotheses which are being assumed or are trivially true.

The crucial parts of the proof — the "idea" of the proof — should be in the arguments at the top of the hierarchy, and the tedious details of the proof should be at the bottom. One can read a proof of this form part way down and be confident that the basic idea of the proof and, hence, the basic idea of the theorem are correct, even though there could be minor problems with the details of the proof and the theorem.

(2) Use modular construction.

Mathematicians have been using modular construction in proofs for centuries, and modular construction is considered part of good programming. It is well understood why modular construction is desirable, even necessary, in mathematical proofs and computer programs. To a large extent the whole enterprise of formal verification rests on one's ability to develop methods of modularity. In conjunction with hierarchical construction, modular construction is an excellent way to satisfy the goal of maintainability.

By making use of proof parts that have been used many times (such as fundamental lemmas), one's doubt in a verification proof can be directed to a few specific aspects of the proof. This helps to increase the reviewer's confidence in the proof by allowing him to concentrate only on what is new. There is some use of modularity in the the RAP verification proof with the use of the theorem prover command logs.

(3) Identify all premises of the proof.

A good proof of any kind should clearly identify all the premises used and assumed in it. Incorrect proofs often result from the use of hidden assumptions that are not valid. Following this suggestion should help in attaining all but the last verification goal. A clear statement of the theorem includes the premises that are assumed (goal 1). In a well-constructed proof, the aspects of the proof which might be questionable are concentrated in the premises of the proof (goal 2). A precise list of premises is a requirement of a rigorous proof (goal 3). Knowing the premises of a proof increases one's understanding of the theorem (goal 4). The premises of a proof identify the conditions under which the proof is valid and can be used again (goal 5).

(4) Use calculations carefully.

Formal verification has been rejected by some [1] as a means of testing the reliability of software. One of the principal reasons for this is that all too often verification proofs contain massive amounts of opaque logical calculations. Calculation is certainly a very valuable aspect of mathematical reasoning. However, if one is going to use calculations in a fundamental way in a verification proof, one needs assurances that the calculations are performed correctly. Until one has these assurances, it is very dangerous to accept the results of calculation without closely examining what was done.

Calculations can be incorrect and, when calculations are opaque (such as arithmetic is in a digital computer), there is no way of easily detecting errors. Complicated calculations should be used in a verification proof only when there is a very high assurance that they will be correct. For example, it is appropriate to use arithmetic calculations performed by a good digital computer or simplifications performed by a well-tested and very reliable simplifier. As prominently mentioned by DeMillo et al. in [1], virtually all formal verifications to date suffer in some degree from excessive reliance on formal logical calculation.

(5) Use an expressive high level language.

It is clear that verifiers can greatly benefit from the use of machines to assist in the development of verification proofs. Using machines necessitates working with formal languages. It is exceedingly hard to get a machine to handle formal languages that have the expressibility of the informal languages used by mathematicians. Consequently, verifiers have usually been tempted into using very simple formal languages. This approach keeps the proof development at such a low level that the verifier and reviewer become lost in a heap of trivia.

Relief can only come with the use of expressive high level languages. Languages of this type are difficult to develop and difficult to program a machine to use, but they allow human beings to think naturally and to make use of the richness of modern mathematics.

4. THE REVIEW AND ACCEPTANCE PROCESS

In mathematics the validation or acceptance of a new result is the outcome of a complex interactive process involving the author, the interested community, and to a lesser extent a technical reviewer appointed by the editor of a journal. Based on our experience of reviewing the RAP, we now discuss how the process of validation ought to occur in the verification of design or program correctness. We believe that many useful analogies between the validation processes in mathematics and in verification can be made. However, while these analogies exist, we still think that the two processes have important differences.

In mathematics (or in other areas such as mathematical economics or mathematical physics), workers in each area of research try to build on or improve published results. They are strongly motivated to understand these results and make sure that they are correct. Mathematical papers are normally structured in a way that permits understanding at many different levels. For example, a specialist reader can take a cursory look at a good mathematical paper and still get some notion of the paper's contents. There are also aesthetic reasons for reading mathematical papers. Researchers read technical papers, not only because they can use the results in their own work, but also because reading them is a pleasurable experience. Generally, papers that are not well written are not immediately received, and the results they claim take longer to be accepted by the community of research workers. Insuring that their papers are read is a powerful reason for authors to write clear as well as interesting papers.

In contrast to the situation in the other mathematical sciences, very few people read verifications of large programs or systems. Most of the reasons that exist for reading research papers do not exist for reading verification proofs. Verification proofs are tedious and rarely provide a basis for further research in the same way as mathematical proofs. Cursory readings of verification proofs provide absolutely no insight. In short, verification proofs are written with no intention of attracting readers. Generally, new software has been accepted exclusively on the claims of developers. In the best of circumstances (as was the case for the RAP) the design or code undergoes some sort of independent review process. Even in the case of the RAP, the review process for the design verification was not explicitly discussed in the original verification plan, despite the fact that a review process for code development was carefully established.

As we have argued above, the existence of proofs, even automated ones, is in itself no guarantee of correctness. Proofs have to be submitted to a thorough review process in much the same way as in mathematics. Since it is unlikely that verifications will attract interested and critical readers, only a formal review process by appointed referees seems to be feasible. This review process should be considered an integral part of the verification effort.

It is our belief that the reviewers should be regarded as the main audience for a verification proof. If the purpose of such a proof is to persuade any potential doubter, then at least the reviewers must be convinced of the correctness of the verification proof. A verification effort which fails to satisfy this condition cannot be considered a proof in any reasonable sense. In order to achieve these goals, the following two conditions at ·st should be met:

Any formal models used in the verification must be understandable without undue effort. Specific guidelines for clarity of specifications should exist to aid specification writers. These guidelines should be agreed upon before any formal models are written.

(2) Even if automated tools are used, the structure of the formal proof must be clearly stated. Presenting the structure clearly makes the automated proof more credible as well as making the verification much more maintainable.

5. SUMMARY

In this paper we have proposed a discipline for verifying software. We think that a verification should consist of three interactive processes: a modelling process, a theorem proving process, and a review and acceptance process. The modelling process should develop formal mathematical models of all requirements, specifications, processes, and systems that are relevant to the verification, and should generate conjectures in the mathematical framework established by the models. The models and conjectures should be clear and their appropriateness justified. In the theorem proving process, proofs of the conjectures generated during the modelling process should be constructed and analysed. Unlike the modelling process, the theorem proving process should be a purely mathematical endeavor. The review and acceptance process should provide a means of communication, so that the verifiers can convince the reviewers — and ultimately the customer — of the adequacy of the verification.

REFERENCES

1. DeMillo, R. A., Lipton, R. J., and Perlis, A. J., "Social Processes and Proofs of Theorems and Programs," *Communications of the ACM* **22** (1979), 271–280.
2. DiVito, B., and Proctor, N., "Restricted Access Processor Formal Security Model," Technical Report TR-82041, Sytek (July 1985).
3. DiVito, B., and Sullivan, E., "Restricted Access Processor System Verification Plan," Technical Report TR-82046, Sytek (October 1983).
4. Halpern, D., and Owre, S., "MUSE: The Sytek Proof Processing System", Technical Report TR-85007, Sytek (July 1985).
5. Lakatos, I., *Proofs and Refutations*, Cambridge University Press, Cambridge, 1976.
6. Levitt, K., Robinson, N. L., and Silverberg, B. A., "The HDM Handbook," SRI International, Menlo Park, California, (June 1979).
7. Proctor, N., "The Restricted Access Processor: An Example of Formal Verification," *Proceedings of the 1985 IEEE Symposium on Security and Privacy*, Oakland California, 1985.
8. Proctor, N., "Restricted Access Processor Message Block Processing System Formal Top-Level Specification," Technical Report TR-83002, Sytek (July 1985).
9. Proctor, N., "Restricted Access Processor Verification Results Report," Technical Report TR-84002, Sytek (July 1985).
10. Scherlis, W. L., and Scott, D., "First Steps towards Inferential Programming", *Proceedings of the IFIP Congress*, Paris, 1983.

THE NATIONAL BUREAU OF STANDARDS MESSAGE AUTHENTICATION CODE (MAC) VALIDATION SYSTEM

Miles Smid, Elaine Barker, and David Balenson

National Bureau of Standards
Institute for Computer Sciences and Technology
Gaithersburg, Maryland 20899

ABSTRACT

This paper describes the National Bureau of Standards MAC Validation System (MVS) for testing the conformance of vendor devices to Federal and commercial data authentication standards. Topics which are covered include the events which led to the development of the MVS, the standards it validates, its design philosophy, the requirements it places on vendors validating their devices, its performance characteristics, and the results of the validations performed to date.

1 INTRODUCTION

In 1979 a group of bankers, vendors, financial network representatives, and a member of the National Bureau of Standards (NBS) met for the first time to define and write an American National Standards Institute (ANSI) standard for authenticating financial transactions. The impetus for the standard came from the bankers who were well aware of the large dollar amounts contained in wholesale electronic financial transactions, and the outdated methods used to protect their integrity.

Two years later, the American National Standard for Financial Institution Message Authentication (Wholesale) was published by the American Bankers Association as ANSI X9.9-1982. The standard made use of the Data Encryption Standard (DES) cryptographic algorithm to calculate a cryptographic checksum or Message Authentication Code (MAC). The originator of a message calculates the MAC by encrypting the data using the DES algorithm and a secret value called the key. The MAC is then sent to the recipient along with the unencrypted message. The recipient, who has the correct key, calculates the MAC in the same manner as the originator and compares it to the received MAC. If the comparison is successful, the data is considered authentic. Otherwise, an unauthorized modification is assumed. Any party trying to modify the data without knowing the key would not know how to calculate the appropriate MAC corresponding to the altered data.

Contribution of the National Bureau of Standards. Not subject to copyright. This research was partially funded by the U.S. Department of Treasury.

The algorithm used to calculate a MAC was based upon the DES cryptographic algorithm which was published by NBS as a Federal Information Processing Standard in 1977 [6]. The International Business Machines Corporation had made the DES specifications available to NBS, and had provided nondiscriminatory and royalty free licensing for building DES devices. NBS established a DES validation program whereby twenty-six DES hardware implementations have been tested for conformance to the DES standard. In addition, the specific method for using DES to calculate the MAC was also published by NBS as Federal Information Processing Standards Publication (FIPS PUB) 113 in 1985 [7].

Much of the ANSI X9.9 standard deals with extraction rules for determining what data in the transmitted message is to be authenticated by the MAC, and editing rules for providing transparency in applications where slight modifications to the data are normally expected. For example, in manual applications the received data must be reentered into the authentication device in order to be authenticated. If even one character is reentered incorrectly or extra spaces are inserted between words, the recalculated MAC will in all likelihood not equal the received MAC. In these cases it may be desirable to minimize the chance of human error by authenticating only the critical fields of the message, and allowing extra spaces to be inserted between words without altering the MAC.

Shortly after the publication of the ANSI X9.9 standard it was submitted to the International Organization for Standardization (ISO) as a candidate international authentication standard. It then became clear that the ANSI X9.9 standard would have to be revised to conform to the character set requirements of the International community. An effort to revise the standard was begun and the revised standard is expected to be published in 1986. Further references to the ANSI X9.9 standard in this paper pertain to the April 7, 1986 version of the revised standard [2].

In 1984, the U.S. Department of Treasury wrote a policy directive requiring that the department's Electronic Funds Transfer (EFT) messages be properly authenticated on all new systems immediately, and on all systems by 1988 [5]. In addition, Treasury decided to certify vendor devices and wrote the criteria that such modules must meet [4].

NBS and the National Security Agency are to assist Treasury with its certification. As a part of this cooperative effort, NBS agreed to develop a MAC Validation System (MVS) which would test conformance with FIPS PUB 113 and ANSI X9.9. This paper will describe the MVS and the tests which are designed to validate conformance to FIPS PUB 113 and ANSI X9.9.

2 MESSAGE AUTHENTICATION STANDARDS

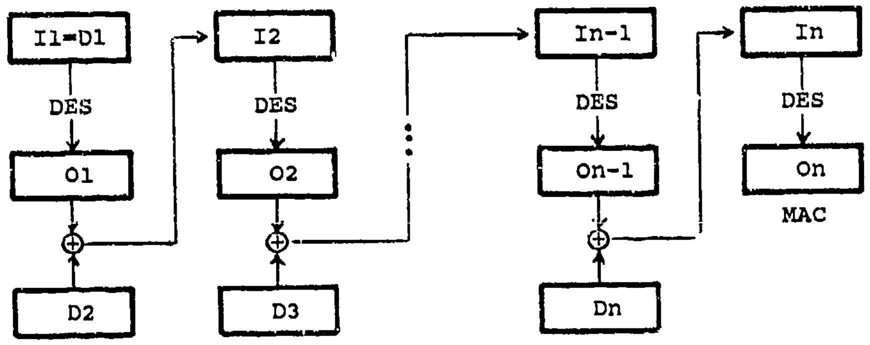
2.1 Computer Data Authentication (FIPS PUB 113)

In automated data processing systems it is often not possible for humans to scan data to determine if it has been modified. Examination may be too time consuming for the vast quantities of data involved in modern data processing applications, or the data may have insufficient redundancy for error detection. Even if human scanning were possible, the data could have been modified in such a manner that it would be very difficult for the human to detect the modification. For example, "do" may have been changed to "do not" or "$1,000" may have been changed to "$10,000". Without additional information the human scanner could easily accept the altered data as being authentic. These threats may still exist even when data encryption is used. It is therefore desirable to have an automated means of detecting both intentional and unintentional modifications to data. Ordinary error detecting codes are not adequate because, if the algorithm for generating the code is known, an adversary can generate the correct code after modifying the data. Intentional modification is undetectable with such codes. However, a cryptographic MAC can protect against both accidental and intentional, but unauthorized, data modification.

FIPS PUB 113 defines an algorithm for calculating the MAC which is consistent with ANSI X9.9 and the Department of Treasury's Electronic Funds and Securities Transfer Policy. The MAC calculation is based on the DES cryptographic algorithm which transforms 64-bit input blocks to 64-bit output blocks using a cryptographic key (See Figure 1). The data to be authenticated is grouped into contiguous 64-bit blocks: D1,D2,...,Dn. If the number of data bits is not a multiple of 64, then the final input block will be a partial block of data, left justified, with zeroes appended to form a full 64-bit block. After the first data block is passed through the DES algorithm the output is exclusive-ORed to the second data block to form the next input to the DES. This process continues until the last data block is exclusive-ORed to a DES output block and the result is used as the last input to the DES. The left-most 32-bits of the final DES output are taken as the MAC.

Since the outputs of each DES transformation are chained to the inputs of the next DES transformation, the final MAC is a function of each bit of data and the secret cryptographic key. When the key is unknown, the alteration of a single bit of data will cause an unpredictable alteration of the MAC. Therefore, any intruder who intercepts authenticated data and attempts to make an alteration does not know what the corresponding MAC for the altered data should be.

The MAC algorithm may be used to protect any data (transmitted or stored) which is exposed to alteration between the initial generation of the MAC and the verification of the received MAC. It does not detect errors which occur before the MAC is originally generated.



Ii	-	64-bit DES input block
Oi	-	64-bit DES output block
Di	-	64-bit message block
⊕	-	bitwise exclusive-OR operation
DES	-	Data Encryption Standard algorithm

Figure 1. The message authentication algorithm

2.2 ANSI X9.9

The ANSI X9.9 standard defines a uniform process to facilitate the protection of wholesale financial messages. The process is independent of the transmission media, can be implemented in both automated and manual systems, and is usable by both large and small financial institutions.

2.2.1 The Authentication Process. Given a message to be transmitted from the originator to the recipient, the authentication process involves three steps.

(1) The originator of a message computes a message authentication code (MAC) from the contents (or selected contents) of the message using a secret key and one of five authentication options provided by the standard. The five authentication options, which are described in more detail below, include one option for binary data and four options for ASCII messages which involve the authentication of selected parts of a message. Choice of the authentication option and key is the responsibility of the originator and the recipient and should be specified using procedures that are part of a bilateral agreement between the originator and the recipient.

(2) The originator transmits both the unencrypted message and its MAC to the recipient of the message.

(3) The recipient verifies the received MAC with the message by computing another MAC from the contents (or selected contents) of the received message (excluding the MAC itself, and its delimiters, if any) using the same authentication option and key used by the originator, and comparing the computed MAC to the MAC received with the message.

The authentication process can be implemented either through software or special hardware devices or a combination of the two. The process provides verification that the contents (or selected contents) of a message have not been accidentally or deliberately modified during transmission between the originator and the recipient. In addition, the identity of the originator of a message is implicitly verified by proper use of the correct secret key. By including the date and a unique message identifier in a message, the authentication process also provides verification of the uniqueness of a message (i.e., that the message is not a duplicate). The message identifier, which must be authenticated, is a value that does not repeat (typically a sequence number), such that there is not more than one message with the same message identifier that has the same date and uses the same key.

The authentication process alone does not guarantee absolute security. The protection provided applies only to the parts of a message that are actually authenticated. Other parts of a message are subject to undetected alterations. Written agreements, physical, personnel, and procedural security controls are necessary for secure implementation, use, and protection of the authentication process and devices. Keys must be protected in accordance with ANSI X9.17 [3].

2.2.2 The Authentication Computation. The MAC computation involves the application of the authentication algorithm to the contents of a message based on the authentication option used. The algorithm is essentially identical to the authentication algorithm defined in FIPS PUB 113.

2.2.3 The Binary Authentication Option. The binary authentication option applies the authentication algorithm to the entire body of a message represented as a sequence of bits. The MAC is placed in the message in a predetermined location according to a bilateral agreement between the originator and the recipient.

The binary authentication option of ANSI X9.9 provides compatibility with FIPS PUB 113 and is the recommended option for the authentication of bulk data.

2.2.4 The Coded Character Set Authentication Options. The four coded character set options apply the authentication algorithm to either the entire contents or selected contents of ASCII messages. All characters of a message must be represented as 8-bit ASCII characters with the leftmost bit set to zero and the right-most seven bits set as defined by ANSI X3.4 [1]. If the message is represented by a different character set (e.g., EBCDIC), then the message must be transformed into ASCII before selecting the contents of a message and computing the MAC.

In all four coded character set options, an ASCII message contains fields (or message elements), which are contiguous strings of characters designated for a specific purpose. Examples of fields that may appear in a financial message include the identities of the credit, debit, and beneficiary parties, the transaction value and currency types, and the identity of the key used for authentication (IDA).

These fields may or may not appear in a message, but they must be authenticated if they do appear. Other fields that must always appear in a message and must also be authenticated include the date of message origination (Date) and a message identifier (MID). A MAC must also appear in a message, but is not included in the MAC computation. The formats of the IDA, Date, MID, and MAC fields are fixed by the standard, and each of these fields must appear only once in a message.

In order to locate and identify the fields in a message they must be either implicitly or explicitly delimited. A field is implicitly delimited if its placement in a message is either fixed or unambiguously specified by format rules. A field is explicitly delimited if its placement in a message is identified by a complementary pair of opening and closing explicit delimiters without any intervening delimiters. The standard establishes the following opening and closing explicit delimiters:

	Open	Close
Date:	QD-	-DQ
IDA:	QK-	-KQ
MAC:	QM-	-MQ
MID:	QX-	-XQ
Text:	QT-	-TQ

Figure 2 depicts a sample financial message which uses these explicit delimiters. The use of implicit delimiters versus explicit delimiters and the formats of fields that are not fixed by the standard should be specified in the bilateral agreement between the originator and the recipient. In all cases, if a message does not conform to these rules, then a syntax error must be indicated.

The differences among the four au thentication options involve which parts of a message are actually authenticated, i.e., which parts are input to the authentication algorithm in order to compute a MAC for the message.

(1) The Entire Message with No Editing Option simply applies the authentication algorithm to the entire message.

(2) The Extracted Message Elements with No Editing Option applies the authentication algorithm only to the message elements and their delimiters.

The two non-editing options are recommended for the authentication of data whenever the transmission medium provides transparency.

(3) The Entire Message with Editing Option applies the authentication algorithm to the entire message, but first edits the contents according to several editing rules which modify carriage returns and line feeds, convert all alphabetic characters to upper-case, delete all but certain acceptable characters, eliminate leading spaces, and compress sequences of consecutive spaces.

(4) The Extracted Message Elements with Editing Option applies the authentication algorithm only to the message elements and their delimiters after editing the contents as above.

The editing options are recommended for the authentication of ASCII data whenever the transmission medium is not transparent to the character set being used (e.g., BAUDOT networks, Telex).

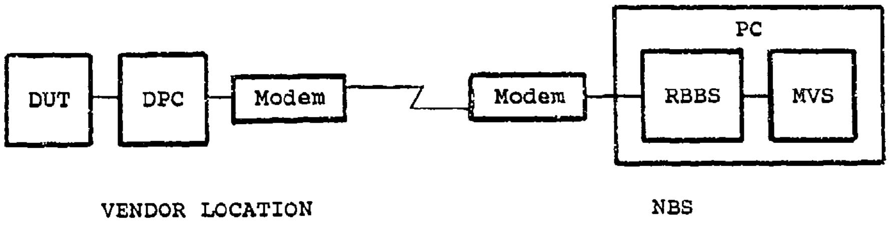
3 MAC VALIDATION SYSTEM

3.1 Design Philosophy

The approach taken in the development of the MVS was based on experience gained from the NBS DES validation process. Costs and staff time to administer the tests had to be kept to a minimum. It was therefore decided that the tests would be automated and performed on test devices at remote locations. Since the tests were to be automated, NBS staff would only have to monitor the results of the tests. And, since the tests could be performed on remote devices, shipping and set-up expenses would be eliminated.

```
TO YOUR BANK

FROM OUR BANK

QD-80 07 14-DQ ///// 1056/  QX-127-XQ

QT-

TRNSFR USD $1234567,89 FRM ACCNT 48020-166
             /////      TO ACCNT 40210-178

-TQ

KEEP ON QT EXPECT VISIT ON FRIDAY OF
NEW DIV VP ON PROJECT QT-QWERT-TQ BE

Careful

REGARDS

QUIRTO
QK-1357BANKATOBANKB-KQ
QM-D21F 3879-MQ
```

Figure 2. Sample financial message



DUT = Device Under Test
DPC = Device Protocol Converter
PC = Personal Computer
RBBS = Remote Bulletin Board System
MVS = MAC Validation System

Figure 3. Basic configuration

When initial DES validations were performed, much time was spent interfacing vendor devices to the NBS test device. In the case of the MVS, it was decided to specify the MVS interface and require that the vendor match the device to the interface. In most cases this can be accomplished with a PC-based device protocol converter (DPC) because the interface is represented as specific message protocols, including message flow and format, between the MVS and the device (see Figure 3).

The intent of the validation process is to provide a rigorous conformance test which can be performed at a modest cost. NBS does not try to prevent a dishonest vendor from purchasing a validated device and remotely validating the device as the vendor's own product. However, customers who wish to protect themselves against a dishonest vendor could require that the vendor revalidate the device in the customer's presence.

3.2 Basic Configuration

The MVS is implemented on a personal computer (PC) equipped with a 1200 baud modem and a DES encryption board. A public domain Remote Bulletin Board System (RBBS) is used to provide controlled access to the MVS by the vendor and the vendor's message authentication device, the Device Under Test (DUT) (see Figure 3). In addition, the RBBS features could be used to provide the user with information on how to use the system and with a list of currently validated products. The MVS is accessed using the "WINDOW" command of the RBBS which allows programs to be run which are external to the RBBS program. When the MVS is activated, the user's identity and password are requested followed by a menu of options for debugging, validating and status checking. Test activity is logged in order to resolve any discrepancies in expected test results.

3.3 Validation Protocol

The MVS permits the vendor to both debug and validate a device for any of the five ANSI X9.9 authentication options (and FIPS PUB 113) using similar protocols (see Figure 4). Details of these protocols, including the specific message flow and formats, are given in the NBS Message Authentication Device Validation Requirements [11]. The debug capability permits the vendor to test the MVS in the same manner that the MVS tests the DUT during validation. This capability is provided for the vendor's benefit and is not required for validation.

During validation the MVS attempts to validate the DUT for each selected option by sending requests to which the DUT must respond. As depicted in Figure 5, validation message flow begins with the DUT sending a READY message to the MVS to indicate that the DUT is ready to proceed

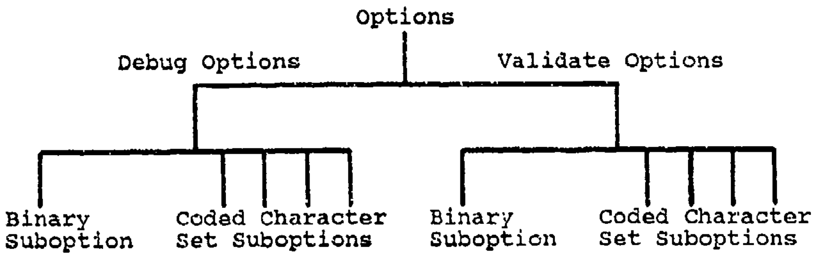


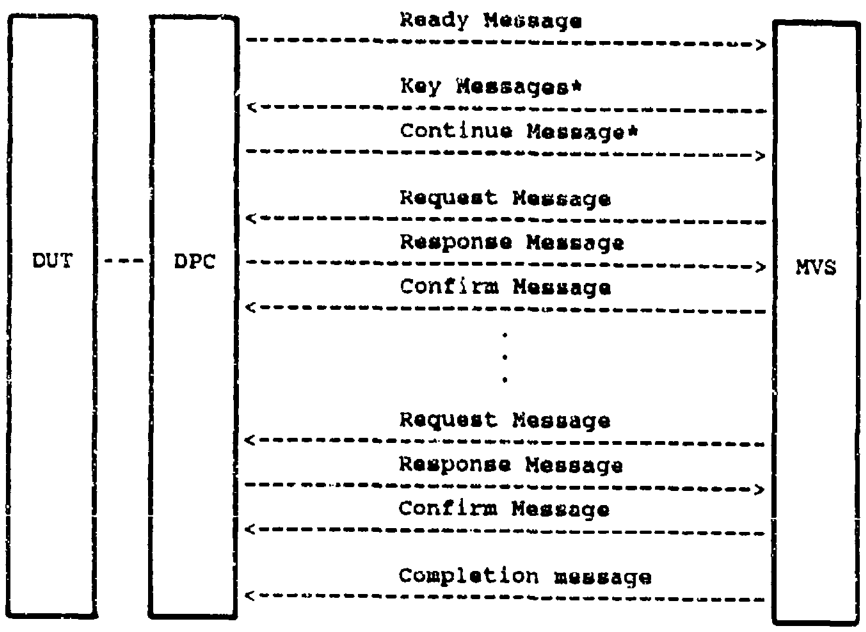
Figure 4. MVS Debug and Validate Options

with testing. Validation testing for the Binary Option will begin at this point; whereas, for each of the Coded Character Set Options, a sequence of ten keys and associated identities must first be sent by the MVS, followed by a CONTINUE message which is sent by the DUT.

A series of validation tests follows, each test consisting of a request message, a response message and a confirm message. A request message is sent by the MVS to request that the DUT either compute a MAC from a message, or verify a MAC in a message. A request message for the Binary Option consists of a key and data pair. The data may or may not contain a MAC. The request message for a Coded Character Set Option contains a sequence of ASCII characters formatted according to the Coded Character Set Rules described in ANSI X9.9. The response message is sent by the DUT and may have several forms, depending on the exact nature of the request message. It may contain the computed MAC of the data contained in the request, an indication of whether or not the MAC contained in the data is correct, or it may indicate that the data had a syntax error. The confirm message is sent by the MVS to indicate whether or not the DUT returned the correct response. Upon completion of a validation option, the MVS sends a completion message to indicate the final status of the testing for that option.

During validation testing the MVS maintains a retest count. Whenever the DUT provides an incorrect response to a test, the MVS automatically repeats the same request in the next test. If the DUT provides the correct response within three tests using the same request, the retest count is incremented by one. If, however, the DUT provides an incorrect response for three tests using the same request, the retest count is incremented by a large value to indicate a test failure. Testing continues in either case. At the conclusion of testing, the retest count is evaluated to determine the validation status. If the retest count is greater than 5 (because of a complete failure on a test or the retest of more than 5 different requests), the completion message indicates that the validation option was not completed successfully. Otherwise, the completion message indicates that the option was successfully completed.

The DUT may receive validation credit for any of the five options. However, successful validation of the Binary Option is a prerequisite for successful validation of any of the four Coded Character Set Options.



DUT = Device Under Test
DPC = Device Protocol Converter
MVS = MAC Validation System

* The key and continue messages are used in the Coded Character Set Options only

Figure 5. Message flow for the validate suboptions

3.4 Validation Tests

3.4.1 Binary Option Tests.

The following types of tests are performed in the Binary Validate Suboption:

(1) 235 selected key and data combinations (without a MAC) which are related to those given in Appendix B of NBS Special Publication 500-20 [8], except that the data consists of the given data with one to eight hexadecimal ASCII ones appended. These tests are used to check for the proper functioning of the DES algorithm.

(2) 192 selected key and data combinations (without a MAC) which are related to those generated by the DES Maintenance Test as specified in NBS Special Publication 500-61 [9], except that the data consists of the generated data with one to eight hexadecimal ASCII ones appended. The DES Maintenance Test creates a cycling process consisting of a maximum of 192 encryption and decryption operations intermixed in such a way as to test all aspects of the DES algorithm. These encryption and decryption operations are used here as an additional check for the proper functioning of the DES algorithm.

(3) At least 100 key and data combinations (without a MAC) which are randomly generated. Some of the combinations consist of data whose length is not a multiple of 64 bits so that the DUT has to correctly pad the data in the MAC computation. These tests are used to check the ability of the DUT to correctly compute a MAC.

(4) At least 100 key and data combinations (with a MAC) which are randomly generated. Approximately half of the MAC's are randomly chosen to be incorrect. These tests are used to check the ability of the DUT to correctly compute a MAC and compare it to a given MAC.

3.4.1 Coded Character Set Options Tests.

For all of the Coded Character Set Validate Suboptions the following are tested:

(1) The ability of the DUT to compute a MAC. The example message given in Appendix B ' ANSI X9.9 is used along with several other test messages. Messages that are modified by deleting, inserting, modifying, and transposing characters are used to check that the DUT can detect such modifications. Messages of varying lengths, and hence, requiring padding are used, as well as messages which include the entire ASCII character set both with and without the parity bits being set.

(2) The ability of the DUT to compute a MAC and compare it with a received MAC. The same messages used above are used here, except that the messages should contain a MAC. Approximately one half of the messages contain an incorrect MAC.

(3) The ability of the DUT to process explicit delimiters. The messages used contain incomplete explicit delimiters, lowercase explicit delimiters, unexpected opening or closing explicit delimiters, missing closing explicit delimiters, mismatched opening and closing explicit delimiters, and pairs of explicit delimiters that are transposed.

(4) The ability of the DUT to process message element formats. Messages containing the message elements with fixed message element formats (i.e., Date, IDA, MAC, and MID) are used. Both correct and incorrect message element formats are checked.

(5) The ability of the DUT to process messages which are missing required message elements or contain multiple occurrences of message elements which should appear only once.

(6) The ability of the DUT to apply the message element extraction rules for the extracted message element options (editing and non-editing).

(7) The ability of the DUT to apply the editing rules in the proper order for the editing options (entire message and extracted message elements). Messages which exercise all of the editing rules are used.

3.5 Validation Procedures

The NBS Message Authentication Device Validation Procedures [10] outline the steps that must be followed by a vendor wishing to use the MVS to validate their message authentication device as part of the Treasury certification process. The procedure consists of six steps, including the application to Treasury, validation by the MVS, and final certification by Treasury.

4 MVS IMPLEMENTATION ISSUES

4.1 Performance Issues

Since validation will be performed from remote locations using dialup access, the telephone lines may introduce errors if a poor connection is obtained. The protocol has been designed to allow for recovery from communication garbles by repeating messages upon request. In addition, the length of time required to conduct testing was chosen to be short enough that a vendor will have a high probability of passing a test if the vendor's device has been correctly implemented, but lengthy enough to test the vendor's device for conformance to the implementation requirements. Using a 1200 baud commmunications line, the test set for the Binary Option, which consists of 627 messages, requires about 21 minutes to validate, whereas the test set for the Coded Character Set Options, which consists of 455 messages, requires about 13 1/2 minutes. Final validation will, therefore, be completed in about 75 minutes if all of the options are tested.

4.2 Problems and Issues Encountered and Solved

A number of problems were encountered during the implementation of the MVS. Some of them along with their solutions were:

(1) During the testing of the Coded Character Set Options, two different responses were required. Sometimes it was desired that the DUT compute a MAC, compare it to a received MAC, and respond with the result of the comparison. At other times, it was desired that the DUT compute a MAC and respond with the computed MAC. As a result, two types of request messages are sent by the MVS for the Coded Character Set Options.

(2) For the Binary option, it was considered desirable to include key and data combinations that would test all functional aspects of the DES algorithm (e.g., permutations and S-boxes). The initial data was taken from NBS Special Publication 500-20 [8]. However, it was found during the first official validation that many of the tests were failing because the test set included self-dual (weak) keys and the DUT rejected these keys. It was therefore decided to modify the tests so as to not include the four self-dual keys.

(3) For the testing of the Coded Character Set Options, ANSI X9.9 does not specify that the IDA (key identity) field is required. In actual operation the users would use a key that was previously agreed upon (a "default" key). It was decided to use the first key of the ten keys sent at the beginning of testing as the "default" key. The question was also raised of how to handle the the case where the IDA delimiters are present, but the field is empty, or NULL. It was decided to handle this as a key whose identity was NULL rather than using the "default" key.

(4) There is a problem inherent in testing several options. How do you prevent a vendor from modifying their device between the successful validation of one option and the testing of another option? The modification could affect the results of the previously successful validation if it were to be rerun. Therefore, a final validation step was included which tests each of the options in sequence as selected by the vendor. Credit for the validation of the vendor's device is not awarded until this final validation process is performed.

(5) For the testing of the Coded Character Set Options, ANSI X9.9 does not specify any bounds on the value of the date field. It does not specify whether a year of 01 refers to 1901 or 2001, nor whether to check the number of days in February, which varies depending on leap years. While the addition of such checks on the date might be reasonable and desirable, without a standard way of doing so there is no way to guarantee that a DUT would implement such a check in the same fashion as the MVS. Therefore, checking the values of the date field was not included in the MVS. It was decided that this type of checking must be performed outside of the authentication process.

4.3 Successful Validations

During May 1986, the Personal Computer Security Module (PCSM), a product of Analytics Communications Systems, Inc., successfully completed the NBS tests for the Binary Option of ANSI X9.9. Other vendors and organizations have expressed an interest in and the intent to validate authentication devices and software using the MVS.

5 FUTURE EFFORTS

NBS plans to continue to support the MVS as a part of the Treasury certification process. In addition, NBS will support the MVS for other Government and commercial applications. It is expected that 6-7 message authentication devices will be validated within the next year.

NBS is beginning to develop the Key Management Validation System (KMVS) which will permit remote conformance testing of the automated key-distribution protocols specified in ANSI X9.17, Financial Institution Key Management (Wholesale). The KMVS will be similar to the MVS, but more complicated due to the complexity of the standard, especially the wide variety of options allowed by the standard.

6 CONCLUSION

NBS has developed the MAC Validation System which is incorporated into a bulletin board system to permit automated remote conformance testing. It was necessary to define protocols interfacing the MVS and DUT in order to allow testing of different vendor devices. This approach reduces the amount of manual intervention and overall costs while providing a rigorous conformance test for the FIPS PUB 113 and ANSI X9.9 standards.

REFERENCES

[1] American National Standard X3.4-1977, Code for Information Interchange, American National Standards Institute, New York, New York, 1977.

[2] American National Standard X9.9-1986 Draft Revision, Financial Institution Message Authentication (Wholesale), American Bankers Association, Washington, D.C., April 7, 1986.

[3] American National Standard X9.17-1985, Financial Institution Key Management (Wholesale), American Bankers Association, Washington, D.C., 1985.

[4] Criteria and Procedures for Testing, Evaluating, and Certifying Message Authentication Devices for Federal E.F.T. Use, Department of Treasury, May 1, 1985.

[5] Department of Treasury Directive 81.80, Electronic Funds Transfer Policy, August 16, 1984.

[6] Federal Information Processing Standards Publication (FIPS PUB) 46, Data Encryption Standard, National Bureau of Standards, Washington, D.C., January 15, 1977.

[7] Federal Information Processing Standards Publication (FIPS PUB) 113, Computer Data Authentication, National Bureau of Standards, Washington, D.C., May 30, 1985.

[8] Gait, Jason, Validating the Correctness of Hardware Implementations of the NBS Data Encryption Standard, NBS Special Publication 500-20, Revised September 1980.

[9] Gait, Jason, Maintenance Testing for the Data Encryption Standard, NBS Special Publication 500-61, August 1980.

[10] Smid, Miles, Elaine Barker, and David Balenson, Message Authentication Device Validation Procedures, National Bureau of Standards, April 15, 1986.

[11] Smid, Miles, Elaine Barker, and David Balenson, Message Authentication Device Validation Requirements, National Bureau of Standards, April 15, 1986.

Using Software Analysis Tools To Analyze The Security Characteristics of HOL Programs

Alan C. Schultz

Computer Science and Systems Branch
Information Technology Division
Naval Research Laboratory
Washington, D. C. 20375-5000
(202) 767-3157
arpanet: schultz@nrl-css

July 17, 1986

ABSTRACT

This paper describes research recently started to discover if existing software analysis tools can be used to find classes of security errors in existing military software. It is assumed that the requirements and specifications for the software are not available, and that only the source code is used in the analysis.

Introduction

The current arsenal of security analysis techniques relies on the fact that the program being analyzed is either under development or has recently been developed, and therefore, that the requirements and the specifications are available to the security analyst. These techniques are of little use in analyzing existing software for which such documentation is unavailable. Without the requirements and specifications available to the security analyst, automated analysis tools that can scan the source code for security flaws would be a useful addition to the security arsenal. The aim of this research is to use existing software analysis tools (e.g. data flow analyzers, flowchart generators, etc) to see if they can detect certain classes of security errors in source code.

Many security flaws in software are the result of poor programming practices or software bugs inadvertently introduced by the programmer. Put another way, many software errors can be exploited as security flaws. By concentrating on these software errors, an analyst can make a classification relating the errors to associated security flaws. These security flaws can then be found by using the appropriate software analysis tool. Although this technique will not identify all security flaws in a program, it will identify many flaws that are associated with software errors.

The goal of this research is to identify classes of security flaws that can and cannot be revealed through the application of software analysis tools. Both static and dynamic analysis techniques will be applied to programs seeded with security flaws to identify these classes. This paper reports preliminary results on tools for static analysis and how they may help analysts locate security flaws.

Analysis Techniques

Software analysis tools can generally be broken into two classes: static analysis tools and dynamic analysis tools. Static analysis tools examine the source code without executing it; dynamic analysis tools analyze the complied code by instrumenting and executing it. In the first phase of this study, static analysis tools will be examined and in the next phase, dynamic analysis tools will be used.

Dynamic analysis is expected to yield more information about the security characteristics of the software, but at a greater cost both in algorithmic complexity and labor than static analysis. Static analysis techniques are more easily automated and, in general, the results need less human interpretation.

Static Analysis Techniques

Static analysis tools can be classified by function: code analysis, program structure analysis, program module interface analysis, and event sequence analysis [20].

Code analysis is a syntactic check of the source code; it is an extension of the compilation process. Several common programming errors can be found with this method, including improper use of variables (e.g. variable used but not initialized, variable initialized but not used) and error prone constructions. Although many newer compilers now check for these errors, older compilers do not, and so this capability is important in the analysis of existing software. Code analysis may also be used to extract information that can be used later for checking the relationships between modules of the program, i.e. parameters, global variables, etc.

Structure analysis can be used to construct graphs of the program which can then be checked for flaws such as improper loop nestings, unreferenced labels, and unreachable statements. Termination checks can be performed in cases where the loop controlling variables are data-insensitive.

Whereas the previous two types of analysis affect single procedures or subroutines, module interface analysis looks for semantic defects across their boundaries. The purpose of this analysis is to detect inconsistencies in the declaration and use of global data structures and parameters. For example, the types and number of parameters should be consistent.

With event sequence analysis, specified events are examined to assure that they are in the proper sequence. For example, in writing to a file, the file must be opened, written to and then closed. Event sequence analysis appears to be the most effective approach to finding security related errors as will be discussed later.

Now that the various functions of static analysis tools have been examined, specific types of analyses will be examined, with reference to specific tools. Most tools combine several of the above functions.

Complexity Analysis

When a security analyst begins to analyze a large system, he needs some method of deciding where to begin. Most systems are too large to desk check in their entirety. One method is to identify the most complex modules, and use those modules as a starting point for further analysis.

One recently developed static analysis tool is based on software complexity metrics. Developed by the U.S. Army Electronic Proving Grounds, the Fortran Complexity Analysis Program (FCAP) [6] calculates McCabe's cyclic complexity [15] and the components needed to calculate Halstead's various metrics [10].

The use this kind of tool in locating security flaws is indirect. As noted by C. R. Attanasio in an operating system penetration report, "...relative design simplicity was found to be the source of greatest protection against penetration efforts. ...simplicity enhances the probability of obtaining security." [2] A security analyst can use a tool such as FCAP to find the most complex modules in the software and use that as the basis for a more complete desk check.

McCabe's metric measures complexity based on how many control paths exist in a single module. If a control graph of a module written in a high level language is created, McCabe's metric would be the number of faces of the graph (regions in a planar graph) plus one. According to McCabe, no module should have a complexity greater than ten. FCAP will identify all modules which have a McCabe complexity greater than a user-defined value. All such identified modules would then be subjected to a more rigorous desk check or further static analysis.

Halstead's metric is an estimation of the length of a module or program, and is calculated by formula using a count of the number of distinct and total operands and operators. Although various tools will calculate Halstead's length metric, this metric does not seem to have the same applicability to locating security flaws in software as McCabe's.

Pattern-directed analysis

Probably the most useful technique in security analysis is pattern-directed analysis. In this technique, a suspected security flaw is characterized as one or more statements in sequence, but not necessarily adjacent. The sequence is then searched for, and if found, is subjected to a desk check.

What patterns are suspicious? "From the software point of view, both the operating system and each application program bear responsibility for maintaining data security. It is, however, the operating system that controls, assigns, allocates, and supervises all resources within the computer system." [1] Various resources and data are accessible to an application program only after "appropriate dialogue (i.e. system calls) with the operating system. ...should the operating system be tricked...or compromised by an application program, the confidentiality of information may be violated." [1] Therefore, one area of interest might be to examine the application program's use of system calls, or any calls outside of the software being examined.

Also suspect are routines that attempt core dumps, routines that do not clear memory buffers or data areas after use, direct addressing of memory, non-documented instructions or instructions with known, undesirable side-effects, etc. For example, all constants other than zero and one in a program should be regarded as suspect since if not used in unit conversions, constants could indicate the use of direct memory access.

The most well-known tools of this class are the RISOS (Research in Secured Operating Systems) tools [19]. Several of the tools in RISOS will search an assembly language program for selected patterns that might indicate security flaws. The security analyst can enter a suspected pattern and either have the number of occurrences of that pattern reported, or have the location of the occurrences flagged. The analyst can also have the lack of some pattern flagged.

The RISOS tools were specifically designed for assembly language analysis. However, the simple pattern-directed search of the RISOS tools is not sufficient for high-level languages. Due to the control structure of high-order languages, two patterns may not appear to follow each other in a simple top-down search. For high-order languages, the control structure must be taken into account. Control flow analysis along with data flow analysis is a technique that allows more robust type of pattern-directed search.

Control Flow Analysis

Control flow analysis examines the control structure of high-level programs. This technique allows checking programs for improper subprogram usage and violations of control flow standards.

By itself, this technique allows a limited number of flaws to be detected. In particular, unexecutable (unreachable) sections of code can be identified, and a call graph of the program and flow graph of each module can be created and manually inspected. Additionally, it is possible to find violations of a specified standard, e.g. backward jumps out of a control structure are not allowed.

The call graph indicates the structure of the program with respect to subroutines and possible errors. The presence of cycles in the call structure indicate recursion, routines that are never called indicate unreachable code, and attempts to call nonexistent routines are flagged. The flow graph can make dead code evident indicating improper use of boolean expressions.

Concerning security, one use for control flow analysis would be to check for trap doors remaining from the debugging of the software. If the control flow analyzer attaches predicate information to the arcs of the graph, these predicates could be examined for comparisons to string constants.

Although by itself the technique finds only a limited range of flaws, the call graphs and flow graphs are essential for data flow analysis.

Data-flow Analysis

Data flow analysis inspects patterns of data use in a program exposing error-prone design and programming practices. Although data flow and control flow analysis are separate techniques, most data flow analysis tools now incorporate some form of control flow analysis to untangle the high-level control structures. In the data flow tools examined, control flow is an essential part of the process.

Data flow techniques were originally used to optimize code generated by certain compilers [17] and was later applied in static analysis of software. This technique has been applied in software validation and documentation of Fortran programs [16], and several tools have been developed [18, 26]. Data flow based tools have also been developed for other languages including PL/1 [23].

Data flow analysis searches for anomalies in the source code. An anomaly exists when a variable is used in a way that is inconsistent with the previous or subsequent uses of that variable in the program.

A typical data flow analysis tool must first parse the source code and generate an internal representation of the program, usually in the form of a tree. Second, a control flow graph of the software is created with attached variable information. This data is used to perform the data flow analysis.

The various tools that utilize data flow analysis differ in the errors they report, but in general, the errors which can be found are:

1) reference to variables not defined or set;
2) variables set but not defined;
3) variables set and not used (or set and then set again without being used between the two settings);
4) all of the errors listed under pattern directed analysis;
5) all of the errors listed under control flow analysis if performed.

It should be noted, however, that the technique will allow any specified sequence of statements to be found. In this respect, data flow analysis holds the most potential in security analysis. The objective is to characterize a security flaw as a sequence of statements, and then search for that sequence.

In this respect, the data flow tools can find the security-related errors reported under pattern-directed analysis. Additionally, patterns that are not obvious due to the control flow of the program may be found. An example of such a pattern is the class of errors characterized by Bisbey as inconsistencies of a single data value over time [4]. In this class of errors, a data value is rendered inconsistent between two operations. More specifically, the data value is changed between pairs of refences. The general pattern specified is:

1) find an operation L which either fetches or stores into a cell X;
2) find an operation M that fetches cell X;
3) operation M is critical (security related);
4) operation L occurs before operation M.

Step 4 requires that the control flow be part of the analysis. Operation L must then be examined to see if it improperly alters X.

Other Techniques

Other static analysis techniques may also prove useful in detecting security related flaws. Cross-reference generators can reveal misuse of variables. Although limited in scope, a careful scrutiny of the cross-reference listing might be beneficial in a security analysis. Global variable misuse, conflicting variables and useless variables are a few of the errors that can be determined with this technique.

Various program statistics can indicate suspicious variables, i.e. variables only used once or used repeatedly. A variable used only once could be an indication of a trap door or a once used debugging tool. Too many uses could indicate misuse of the variable. The RISOS tools contained a program to analyze the statistics of the module under examination specifically for the above reasons [9].

Limitations of Static Analysis

As stated earlier, although static analysis is more easily automated than dynamic analysis, limitations exist. These limitations can usually be overcome by using dynamic analysis techniques.

The most obvious deficiency is the inability to fully analyze dynamic data types. Pointers and array variables are currently difficult to handle correctly due to their dynamic nature. Current static analysis tools can only treat an array as a single variable, since they cannot know the bounds of the array in some languages. In some cases, pointers cannot be treated at all. The indices of arrays and the objects of the pointers may not be known until execution. Desk checks must still be performed on the code to analyze the use of dynamic data types.

Another deficiency is the handling of recursive and concurrent procedures. Some of the control flow analysis and data flow analysis techniques can follow recursion to a pre-defined level, but only at an enormous cost in resources. Concurrent processing is a current research area in static analysis, but the technology has not yet filtered down to available tools. [25]

Specific Tools

Various tools currently available through government or industry will perform the analyses discussed above. While many of the tools are still experimental in nature, others are proving to be useful production tools in analyzing software for bugs.

FCAP, briefly discussed above, is a tool based on complexity metrics. In addition to calculating McCabe's metrics and calculating the values needed for Halstead's metrics, FCAP will calculate additional metrics, i.e. number of comment lines, number of executable lines, number of entry and exit points, number of forward and backward branches, number of conditional branches and more. FCAP will also produce structure diagrams of each procedure, variable usage report, calls report (all calls from each procedure within a module), and an undefined external variables report.

The tool is written in Fortran and analyzes Fortran source code (VAX, PDP-11, SKU Fortran and RATFOR). It is interesting to note that the tool was used to test itself. In addition to FCAP, the U.S. Army Electronic Proving

Ground has also written similar tools to analyze C and various assembly languages.

Weiser's Data Flow Slicer is an experimental tool that performs data flow analysis on Fortran programs [26, 27]. The program will create a data flow "slice" on each variable in a write statement. These slices will contain all statements which affect the variable being sliced on, essentially exposing the data flow to a variable.

RXVP is a comprehensive, production tool by General Research Corporation which provides static and dynamic analysis for large Fortran programs [22]. This tool performs syntax and structural analysis to detect inconsistencies in program structure and use of variables. The tool generates call graphs, cross-reference listings, variable usage reports (set, used, set and used), and I/O reports (shows all I/O statements).

The above survey gives a sampling of the various functions available in static analysis tools. Many other tools are available which perform similar functions [3, 5, 8, 11, 12, 13, 14, 28, 24, 23, 21, 18].

Plans

In the next phase of this research, various static analysis tools will be applied to programs seeded with security flaws. The sample programs will be medium-sized military application programs written in Fortran. Fortran was picked since many of the tools analyze that language. Also, sample programs written in Fortran are more readily available.

The result of this effort will be a classification of the types of security errors that can be found using static analysis tools, and, equally important, a classification of those security errors that cannot be found with these tools.

Next, dynamic analysis tools will be investigated and classifications will again be made. Further research may extend the existing tools to create a set of tools whose specific function will be to analyze the security characteristics of software.

Conclusion

Since there is a large body of existing military software that cannot reasonably be subjected to formal proof, applying analysis tools to this software can help assure that the software is free of some classes of exploitable security flaws. The technique may also prove useful in obtaining a B1 "Orange Book" rating from the National Computer Security Center. The B1 rating requires (section 3.1.3.2.1) that the source code be subjected to "thorough analysis and testing" to uncover security flaws [7].

Further research is needed to extend the existing tools in order to remove deficiencies and to make the tools security-specific. With these techniques, some assurance can be made about the security characteristics of a program.

References

1. Abbott, R. P., Chin, J. S., and Donnelley, J. E., "Security Analysis and Enhancements of Computer Operating Systems," NBSIR 76-1041 (April 76).
2. Attanasio, C. R., Markstein, P. W., and Phillips, R. J., "Penetrating an Operating System: A Study of VM/370 Integrity," *IBM Systems Journal* **15**(1), pp. 102-116 (1976).
3. Berns, G. M., "Analysis Tool Tracks Down Bugs in FORTRAN Code," *Computer Design* **24**(6), pp. 169-174 (June 1985).
4. Bisbey, Richard, Popek, Gerald, and Carlstedt, Jim, *Inconsistency of a Single Data Value Over Time*, USC/Infomation Sciences Intitute (Feb 1975).
5. Carre, B. A., "Software Tools for Static Analysis and Formal Verification," in *IEE Colloquium on Computer-Aided Software Development*, IEE, London (April 1984).
6. Defense, Department of, "FCAP, Fortran Language Code Analysis Program : Technical User Manual," U.S. Army Electronic Proving Ground, Ft. Huachuca, AZ (July, 1985).
7. Defense, Department of, "Trusted Computer System Evaluation Criteria," CSC-STD-001-83 (August 15, 1983).
8. Frankl, Phyllis G. and Weyuker, Elaine J., "A Data Flow Testing Tool," pp. 46-53 in *Soft Fair, A Second Conference on Software Development Tools, Techniques, and Alternatives: Proceedings*, IEEE Computer Society, New York (December 1985).
9. Frickel, William G., *RISOS Analytic Tool Description Manual, Part 1: Program Description*, Lawrence Livermore Laboratory, Livermore, Calif. (May 1975).
10. Halstead, M. H., *Elements of Software Science*, Elsevier North-Holland, New York (1977).
11. Hlotke, John R., "Complexity Analysis and Automated Verification," pp. 80-84 in *Conference on Software Tools: Proceedings*, IEEE Computer Society, New York (April 1985).
12. Janusz, Paul, "Application of Software Test Tools to Battlefield Automated Systems.," DTIC #ADA144270 (7/84). Army ARDC
13. Johnson, W. L. and Soloway, Elliot, "PROUST: An Automatic Debugger for Pascal Programs," *Byte* **10**(4), pp. 179-190, Yale University (4/85).
14. Korel, B. and Laski, J., "A Tool for Data Flow Oriented Program Testing," pp. 34-37 in *Soft Fair, A Second Conference on Software Development Tools, Techniques, and Alternatives: Proceedings*, IEEE Computer Society, New York (December 1985).
15. McCabe, T. J., "A Complexity Measure," *IEEE TSE* **SE-2**, pp. 308-320 (Dec. 1976).
16. Miller, E. and Howden, W. E., *Tutorial: Software Testing & Validation Techniques*, IEEE Computer Society, Long Beach, CA (1978).
17. Muchnick, S. S. and Jones, N. D., *Program Flow Analysis: Theory and Applications*, Prentice-Hall, Englewood Cliffs, NJ (1981).
18. Osterweil, L. J. and Fosdick, L. D., "DAVE - A Validation Error Detection and Documentation System for Fortran Progrm for Fortran Programs," *Software-Practice and Experience* **6**, pp. 473-486 (Oct.-Dec. 1976).
19. Project, RISOS, "Handbook for Analyzing the Security of Operating Systems," DOD S5-2068 (Nov. 1976).
20. Ramamoorthy, C., "Testing Large Software with Automated Software Analysis System," *IEEE TSE* **SE-1**(1) (1/75).

21. Robinson, P. J., "A User's view of the SAP-A New Software QA Tool for FORTRAN Programs," pp. 67-71 in *Software Engineering. Proceedings of ESA/ESTEC Seminar (ESA-SP-199)*, European Space Agency, Paris (1983).

22. Saib, S. H., "RXVP: Today and Tomorrow (Program Testing Tool)," pp. 103-125 in *Software Validation, Inspection-Testing-Verification-Alternatives. Proceedings*, ed. H. L. Hausen, North Holland, Amsterdam (Sept 1983).

23. Sarraga, Ramon F., "Static Data Flow Analysis of PL/1 Programs with the PROBE System," *IEEE TSE* **SE-10**(4), pp. 451-459 (July 1984).

24. Steffen, Joseph, "Experience with a Portable Debugging Tool," *Softw. Prac. Exper.* **14**(4), pp. 323-334, Bell (4/85).

25. Taylor, Richard N. and Osterweil, Leon J., "Anomaly Detection in Concurrent Software by Static Data Flow Analysis," *IEEE TSE* **SE-6**(3), pp. 265-278 (May 1980).

26. Weiser, Mark, "Program Slicing," *IEEE TSE* **SE-10**(4), pp. 352-357 (July 1984).

27. Weiser, Mark, "Programmers Use Slices When Debugging," *CACM* **25**(7), pp. 446-452, Univ. Maryland (1982).

28. Wilson, Cindy and Osterweil, Leon J., "Omega - A Data Flow Analysis Tool for the C Programming Language," *IEEE TSE* **SE-11**(9), pp. 832-838 (Sept 1985).

Interpretation of the Bell-LaPadula Model in Secure Xenix

G.L. L[illegible]nbaugh, V.D. Gligor†, L.J. Dotterer, C.S. Chandersekaran, N. Vasudevan

IBM Corporation
708 Quince Orchard Road
Gaithersburg, MD 20878

Abstract

In this paper we review the Bell-LaPadula model for secure systems, which includes the definition of states, state transitions and axioms (properties). The interpretation of the model states and state transition in Secure Xenix is defined, and the access control mechanisms of Secure Xenix are shown to satisfy the Bell-LaPadula axioms. The discretionary security and the activation axioms of Secure Xenix are a superset of those defined in the Bell-LaPadula model.

1. Introduction

We define the interpretation of the Bell-LaPadula security model [Bell76] in Secure Xenix [Gligor 86]. The interpretation explains how the protection mechanisms of the Secure Xenix TCB implement the model. Since the description of the Bell-LaPadula model is formal, and since the Bell-LaPadula model is proven sufficient to enforce a specific DoD security policy, the interpretation of the model in Secure Xenix represents *prima facie* evidence that the design of the Secure Xenix TCB follows that policy.

The interpretation of the Bell-LaPadula model and the access control mechanisms of Secure Xenix are shown to satisfy the model's axioms. After that, one only needs to demonstrate that the individual kernel-call specifications (i.e., kernel DTLSs) preserve the ss-, *-, ds-properties, compatibility, tranquility, and activation properties under the defined interpretation. The definition of the (Bell-LaPadula) model interpretation is required for B2-secure systems. In addition to the interpretation, the demonstration that the individual kernel DTLS/FTLS preserve the above-mentioned properties would be required for B3/A1 secure systems (i.e., "A convincing argument shall be given that the DTLS is consistent with the model" c.f. [TCSEC 83] p. 39).

In section 2 of this paper we review the formal definition of the Bell-LaPadula model including the notions of system state, state transitions, model axioms, secure system state, and secure systems. In section 3 we define the Secure Xenix interpretation of the model and show that the access control mechanisms of Secure Xenix satisfy the model axioms. Section 4 contains the conclusion, and section 5 contains the references.

† Xerox is a trademark of Microsoft

† V.D.Gligor's permanent address is: Department of Electrical Engineering, University of Maryland, College Park, MD 20742

2. Review of the Bell-LaPadula Model

The Bell-LaPadula model is a "state transition" model. That is, the model defines formally system states and rules (actions or operations) that move the system from state to state. Furthermore, the model includes four axioms that must also be preserved by every state and by applications of rules to system states.

2.1. System Stat

A *system state* v is an element of the set $V = (B \times M \times F \times H)$ that is defined below.

B is the set of *current accesses* and is a subset of the set $(S \times O \times A)$, where S is the set of subjects, O is the set of objects and A is the set of access privileges (modes) defined in the system. The set B defines the access privileges each subject has to each object currently.

M is the *access matrix*. It consists of elements $M_{ij} \in A$ that define the set of access privileges subject i may have to object j.

F is a three-component *security function*; the first component, f_s, assigns a maximum security level (clearance) to each subject, the second component, f_o, assigns the security level (classification) to each object; and the third component, f_c, assigns the current security level of each subject. Note that $f_s \geq f_c$.

H is the *current object hierarchy*. It is a subset of all functions $\underline{H}$ from objects O to the power-set of objects O, PO, subject to the following two restrictions:

(1) $O_i \neq O_j \Longrightarrow H(O_i) \bigcap H(O_j) = \Phi$, and

(2) there does not exist a set $\{O_1, O_2, \ldots, O_w\}$ of objects such that $O_{r+1} \in H(O_r)$, $1 \leq r \leq w$, and $O_{w+1} \equiv O_1$.

(The above two conditions imply that the current object hierarchy is a collection of rooted, directed trees and isolated points. They rule out objects with multiple parents at different levels, and cycles. If H is a tree structure, then O_R is an object called the root for which $H(O_R) \neq \phi$ and $O_i \in H(O_R)$ for any $O_i \in O$. Furthermore, O_i is a superior of O_j if $O_j \in H(O_i)$.

2.2. State Transitions

The system transition from state to state is defined by a set of rules (operations) that are requested by subjects on system states. A *rule* is a function that specifies a decision (output) and a next-state for every state and every request (input). Thus, a rule ρ is defined as:

$$\rho: \ R \times V \rightarrow D \times V, \text{ where}$$

$R \times V$ is the set of request-state pair (input) defined in the system for every request, and $D \times V$ is the set of decision-state pair output defined in the system for every request. R is the set of request invocations defined, and D is the set (Yes, No, ?, Error) of request outcomes. "Yes" ("No") means that the request has (not) been executed. The "?" outcome means that any other exceptional condition detected during the application of the rule ρ (e.g., table overflow, etc.).

Let $\{\rho_1, \ldots, \rho_s\}$ be a set of rules. The relation W is set of *state transitions* and is defined for any $R_k \in R$, $D_m \in D$ and $V^* \in V$ by:

$$(R_k,\ D_m,\ V^*,\ V) \in W_{(w)} \text{ iff } D_m \neq ?,\ D_m \notin \text{ Error, and}$$

$$(D_m,\ V^*) = \rho_i(R_k,\ V) \text{ for a unique } i, \quad 1 \leq i \leq s;$$

2.3. Systems, System Appearance, System Actions

Let T be the set of positive integers. X is defined as the set of all request sequences, namely the set of all functions from T to R; Y is defined as the set of all decision sequences, namely the set of all functions from T to D; Z is defined as the set of all state sequences, namely the set of all functions from T to V.

A *system*, $\sum(R, D, W, z_o)$, is a subset of $X \times Y \times Z$ such that $(x, y, z) \in \sum(R, D, W, z_o)$ if and only if $(x_t, y_t, z_t, z_{t-1}) \in W$ for each $t \in T$, where z_O is the initial state. [Note that $x_t(y_t, z_t)$ are individual elements of sequences $x(y, z)$.]

A *system appearance* is defined as each triple (x, y, z) such that $(x, y, z) \in \sum(R, D, W, z_o)$, $x \in X$, $y \in Y$, $z \in Z$.

A *system action* is defined as each quadruple $(x_t, y_t, z_t, z_{t-1}) \in W$, where x_t, y_t, z_t are the t-th request, decision, and state in the sequences $x \in X$, $y \in Y$, $z \in Z$.

Alternatively, $(R_i, D_j, v^*, V) \in R \times D \times V \times V$ is an action of $\sum(R, D, W, z_o)$ iff there is an appearance $(x, y, z) \in \sum(R, D, W, z_o)$ and some $t \in T$ such that $(R_i, D_j, v^*, v) = (x_t, y_t, z_t, z_{t-1})$.

2.4. Model Axioms

The axioms of the Bell-LaPadula model require the definition of the access privilege set A. In the model, $A = \{$ read, write, execute, append $\} = \{r, w, e, a\}$. The meaning of these privileges is defined in the model in terms of the ability to "observe" or "alter" the state of objects and subjects as follows:

e (execute) access = neither observation nor alteration
r (read) access = observation with no alteration
a (append) access = alteration with no observation
w (write) access = both observation and alteration.

The first two of the four axioms (also called "properties" in Bell [76]) use the above access privilege definitions. An additional axiom, called the "tranquility principle," is defined in [Bell 73]. This axiom has been removed from [Bell 76].

2.4.1. The Simple Security (ss) Property

A system state $v = (b, M, f, H)$ satisfies the *ss-property* iff, for each element $b \in B$ that has an access privilege of *read* or *write*, the maximum clearance of the subject dominates the classification of the object; or alternatively:

An element $(s, o, \underline{x}) \in B$ satisfies the *ss-property* relative to the security function f iff

(i) $\underline{x} = e$ or a, or

(ii) $\underline{x} = r$ or w and $f_s(s) \geq f_o(o)$.

The above two definitions restrict the subject access to objects based on object classifications and subject maximum clearances whenever subject accesses to objects include "observation" of the object state. Also note that the *ss-property* restricts subjects from having direct access to information for which they are not cleared.

2.4.2. The *-Property

A system state $v = (b, M, f, H)$ satisfies the **-property* relative to the set of subjects $S' \subset S$ iff, for each element $(s, o, \underline{x}) \in B$:

(i) $\underline{x} = a \Longrightarrow f_c(s) \leq f_O(o)$

(ii) $\underline{x} = w \Longrightarrow f_c(s) = f_O(o)$

(iii) $\underline{x} = r \Longrightarrow f_c(s) \geq f_O(o)$; where S' is the set of untrusted subjects.

The above property is intended to prevent unauthorized flow of information from higher security levels to lower ones In particular, the **-property* prevents an untrusted subject from having simultaneously privileges to "observe" information at some level and to "alter" information at a lower level, namely $[(s, o_i, a), (s, o_j, r) \in B] \Longrightarrow f_O(o_i) \geq f_O(o_j)$. This property represents a restatement of the *-property, and is used as the *-property definition in [Feiertag 77]. Note that, *trusted subjects* (i.e., subjects not in S') need not be bound to the *-property relative to S'.

2.4.3. Discretionary Security (ds) Property

A system state $v = (b, M, f, H)$ satisfies the *ds-property* iff, for every element $(s_i, o_j, \underline{x}) \in B$, $\quad \underline{x} \in M_{ij}$.

2.4.4. Compatibility Property

The object hierarchy H maintains compatibility iff, for any $O_i, O_j \in O$ and $O_j \in H(O_i)$, $f_o(O_j) \geq f_o(O_i)$. This axiom is also called the "nondecreasing level" axiom for the object hierarchy.

2.4.5. Tranquility Principle

The original version of the Bell-LaPadula model [Bell 73] also contained the "tranquility" principle. This principle (axiom) states that a subject cannot chai ge the security level of active objects. Of course, this is defined relative to the untrusted subjects S'.

This axiom has been removed from the 1976 version of the Bell-LaPadula model to allow controlled changes of security levels of active objects. The rules that control such changes depend on specific applications (e.g., mail, guards, etc.) and differ from system to system.

2.4.6. Activation Axioms

Object activation/deactivation refers to the creation and destruction of objects. The dynamic creation/destruction of objects in the Bell-LaPadula model would cause the domain of the classification function f_o and the size of the access matrix M to vary dynamically. To avoid this, the entire set of objects ever used are considered extant in either active or inactive form. Furthermore, objects are considered to be labeled in both forms [Bell74].

However, the use of the above convention requires the specification (1) of a subject's access to an inactive object, (2) of the state of newly activated objects, (3) of the classification of newly-activated objects, and (4) of the object deactivation rules. Specification (1) is necessary because active and inactive objects are assumed to coexist in O. Since the model defines subjects' access to active objects, it must also define subjects' access, or lack thereof, to inactive objects. If left unspecified, such access may cause security breaches in real implementations. Specification (2) is necessary because inactive objects have states (since they exist in O). Thus, their activation must specify the relationship between the state of an inactive object and its state at activation. Similarly, specification (3) is necessary because inactive objects also have a classification in the model, and their classification while inactive might "not match the requirements of the requesting subjects" [Bell 74]. Furthermore, their classification may conflict with the compatibility axiom [Bell 76]. Specification (4) is also necessary because the object deactivation (destruction) rules are security relevant. [As shown in section 3.1.7 the destruction of upgraded directories may not take place at the level where they are read or written.]

Feiertag, Levitt and Robinson [Feiertag 77] attribute two activation axioms to Bell-LaPadula [Bell 74] that specify only a subject's access to one inactive object and the state of a newly-activated object. The two activation axioms are:

(i) *Non-accessability of Inactive Objects* - A subject cannot read the contents of an inactive object; and

(ii) *Rewriting of Newly Activated Objects* - A newly activated object is given an initial state that is independent of the state of any previous incarnations (*n.a.*, activations) of the object. (*n.a.*, This axiom implies that the "object reuse" requirement of [TCSEC 83] is satisfied.)

The two activation axioms can be expressed succinctly as:

(i) Let $O = O' \cup O''$ where $O'(O'')$ = active (inactive) objects and $O' \cap O'' = \Phi$

$$[\forall(s,o,\underline{x}) \in B,\ o \in O''] \Longrightarrow (\underline{x} \neq r \text{ and } \underline{x} \neq w)$$

(ii) Let new(o) = $\{O'' := O'' - o$ and $O' := O' + o\}$ and CALL[S_i, new(o)] be the invocation of the primitive "new" by S_i;

CALL[Si, new(o)] $\Longrightarrow$ state[new(o)] $\neq$ g[state(o)] for any function g and state(o).

2.5. System Security

A state sequence $z = (z_1, \ldots, z_s)$ is a *secure state sequence* iff z_t is a secure state for each $t \in T$.

A system appearance $(x,y,z) \in \sum(R,D,W,z_o)$ is a *secure appearance* iff z is secure state sequence.

A system $\sum(R,D,W,z_o)$ is a *secure system* iff every appearance (x,y,z) is a secure appearance.

An equivalent definition of secure systems can be given by stating that a system satisfies the first three security axioms, namely the ss-property, the *-property, and the ds-property. The following three theorems form the basis for an alternate definition of secure systems.

Theorem A1.

The system $\sum(R,D,W,z_0)$ satisfies the ss-property for any initial state z_0 that satisfies the ss-property iff W satisfies the following conditions for each action $[R_i, D_j, (b^*, M^*, f^*, H^*), (b, M, f, H)]$:

(i) each $(S,O,\underline{x}) \in b^* - b$ satisfies the ss-property relative to f^*; and

(ii) each $(S,O,\underline{x}) \in b$ that does not satisfy the ss-property relative to f^* is not in b^*.

Theorem A2.

A system $\sum(R,D,W,z_0)$ satisfies the *-property relative to $S' \subset S$ for any initial state z_0 that satisfies the *-property relative to S' iff W satisfies the following conditions for each action $[R_i, D_j, (b^*, M^*, f^*, H^*), (b, M, f, H)]$:

(i) for each $S' \subset S$, any $(S,O,\underline{x}) \in b^* - b$ satisfies the *-property with respect to S'; and

(ii) for each $S' \subset S$, if $(S,O,\underline{x}) \in b$ does not satisfy the *-property relative to S', then $(s,o,\underline{x}) \notin b^* - b$.

Theorem A3.

A system $\sum(R,D,W,z_0)$ satisfies the ds-property iff the initial state z_0 satisfies the ds-property and W satisfies the following conditions for each action $[R_i, D_j, (b^*, M^*, f^*, H^*), (b, M, f, H)]$:

(i) if $(S_k, O_\ell, \underline{x}) \in b^* - b$, then $\underline{x} \in M^*_{k,\ell}$; and

(ii) if $(S_k, O_\ell, \underline{x}) \in b$ and $\underline{x} \notin M^*_{k,\ell}$, the $(S_k, O_\ell, \underline{x}) \notin b^*$.

The proofs to the above three theorems can be found in [Bell76, pp. 89-94].

Corollary A1 [Basic Security Theorem].

A system $\sum(R,D,W,z_0)$ is a secure system iff z_0 is a secure state and W satisfies the conditions of theorems A1, A2, and A3 above.

Theorems A4-A6 and A7-A9 of [Bell76, pp. 94-97] represent restatements of Theorems A1-A3 focusing on (1) properties of sets of *system actions* of W, and on (2) properties of *individual states* of V, respectively. In contrast, Theorems A1-A3 focussed on properties of the *current access sets* of B. Similarly corollaries A2 and A3 are the corresponding restatements of corollary A1. Theorem 10 restates the results of the Theorems A1-A3, A4-A6, and A7-A9 in terms of property-preserving rules ρ.

The need for the alternate, but equivalent theorems, becomes apparent when one needs to construct proofs of real systems. For example, in systems whose kernel enforces security, it is substantially more convenient to prove Theorems A4-A6 or A10 than Theorems A1-A3 or A7-A9. The reason is system actions or rules can be easily identified with kernel calls and their effects on the system states.

3. The Interpretation of the Bell-LaPadula Model

The interpretation of the Bell-LaPadula model in Secure Xenix consists of a description of the notion of system state, and state transition in Secure Xenix. Furthermore, it includes the definition of the initial state and an argument that explains why the mandatory and discretionary access control of Secure Xenix implies that the axioms of the Bell-LaPadula model are satisfied.

3.1. The Interpretation of the System State

The interpretation of the system state requires the identification of the state components $B = S \times O \times A$, M, F, and H in Secure Xenix.

3.1.1. Secure Xenix Subjects (S)

Processes are the only type of subject in Secure Xenix. A process may create and destroy objects, may activate and deactivate them, may change the discretionary privileges of objects in the access matrix, may change the current access set, and may change the object hierarchy. However, processes may *not* change the security level of objects. All changes a process makes to the system state are constrained to satisfy compatibility, tranquility, ss-property, *-property, and ds-property. This is discussed in detail below.

Processes are created at login time or by other processes. A process is identified by a unique process identifier and its user is identified by a non-reusable UID and GID [Gligor86]. The effective UID and GID of a process are used in all discrete unary access control decisions. Each process contains a security label that is used in mandatory access control decision. Process labeling is discussed below in the section describing the interpretation of the security function in Secure Xenix, and the use of the real and effective UID and GID is discussed in the interpretation of discretionary access control.

3.1.2. Secure Xenix Objects (O)

The user-created objects of Secure Xenix are: files, special files (devices), directories, pipes, message queues, semaphores, shared memory segments, Xenix semaphores, Xenix shared data segments, ACLs, and processes. Secure Xenix also includes system-created and maintained objects such as the special files (devices) that can be opened or closed by user processes. Trusted processes create, maintain, and use similar objects as those of the users.

(1) *Files, Special Files, Pipes, Xenix Semaphores, Xenix Data Segments, and ACLs*

Files are containers of information managed by the Secure Xenix kernel. Files are protected by either ACLs or by protection bits associated with file i-nodes. The security label of each file is represented in its i-node.

The special files are used to represent devices and can be opened or closed by user processes. In the case of special files the object activation and deactivation are equivalent to the opening and closing of a device. In all other aspects the special files function as the user-created files.

The Xenix shared data segments have similar function to that of the files and are represented, protected, and labeled in a similar way. The difference is that the shared data segments allow asynchronous processes to synchronize their read and write accesses to segment data, and that, unlike files that are shared on a per-copy basis, shared data segments are shared on a per-original basis.

Named pipes function as "unbounded" communication buffers and are represented, protected, and labeled in a similar way as the files. The difference between named pipes and shared data segments is that named pipes impose producer-consumer process synchronization to prevent underflow conditions.

Semaphores are objects that allow the synchronization between asynchronous processes and have similar representation, protection and labeling to that of files.

Access Control Lists (ACLs) are objects used for the discretionary protection of files [Gligor86] and are represented as specially-protected files by the kernel. Each ACL is associated with its file uniquely for the lifetime of the file. The association is maintained by the kernel. The ACLs are labeled with the same label as that of the file they protect. They are discussed in detail in the section on access matrix representation.

(2) *Directories*

Directories are containers for files, special files, pipes, Xenix semaphores, Xenix Data Segments, ACLs, and other directories. They form the building blocks for the system hierarchy. Directories are maintained and protected by the Secure Xenix kernel and are represented in a similar way to that of files. The directories that contain special files and ACLs are system created/destroyed whereas the rest of the directories are created and destroyed by users. A directory that contains an object is called a *parent* directory. A special directory called the root is the highest directory in the parent chain. It is its own parent. It has no ACL and is always "search"-able by all users.

(3) *Message queues, Semaphores, Shared Memory Segments, and Processes*

The objects in this group do not have file system representation. The System V semaphores and shared memory segments have the same function as their Xenix correspondents. The message queues are containers for messages and are used primarily for requests to server processes. Processes are created and destroyed by their parent processes and are identified, labeled, and protected in the same way as that used for their parents.

All objects mentioned above are activated when they are created and deactivated when they are destroyed. Exceptions to this rule are the special files, which activated when they are opened and deactivated when they are closed. Special files (devices) cannot be created/destroyed by users. This is important in the interpretation of the activation axiom (viz. section 3.7).

3.1.3. Access Privilege Set of Secure Xenix (A)

The basic set of access privileges in Secure Xenix consists of the read, execute, write, and null privileges. (An additional privilege, setuid-gid, is defined for executable files. This privilege is discussed in section 3.3 below). These privileges are visible to the user and are interpreted by the kernel differently for different objects. Thus, the actual privilege set is substantially larger than the basic set above. In this section we define the access privileges for each type of object of Secure Xenix and its relationship with the access privileges (modes) of the Bell-LaPadula model.

In examining the relationship between the Bell-LaPadula model privileges and the Secure Xenix privileges it should be noted that the e (execute) privilege of the model does not have any correspondent in Secure Xenix (nor in other systems [Bell76, footnote on p.11]). Similarly, the null privilege of Secure Xenix is not explicitly represented in the model. Furthermore, some of the model privileges have no meaning for some of the Secure Xenix objects and have no representation among the privileges define for those objects. (These cases are denoted by the phrase "no meaning" in the correspondence tables below). Other model privileges that have no meaning for some Secure Xenix objects have representation among the access privileges for those objects, however the access authorization mechanisms ignore their representation. This means that none of the operations defined on those objects may be authorized by the ignored privileges. (These cases are denoted by the phrase "ignored" in the privilege correspondence tables below.)

(1) *File Access Privileges*

read (r) A process granted read access to a file can execute instructions that cause data to be fetched (read) from the file into processor or memory registers that can be manipulated (e.g., copied) by users. The read access of the Bell-LaPadula model maps directly into the Secure Xenix read.

write (w) A process granted write access to a file can execute instructions that cause data in the file to be modified. This access privilege differs from the write access in the Bell-LaPadula model in the sense that it does not allow any observation of the state of the file being modified. The append (a) privilege of the Bell-LaPadula model, maps into the Secure Xenix write privilege. Note that the Secure Xenix write privilege is also necessary for append operations to files. The write (w) privilege of the Bell-LaPadula model maps into the read and write privilege combination of Secure Xenix.

execute (x) A process granted the "execute" (x) privilege to a file can transfer control to that file and cause portions of the file to be interpreted and executed as instructions. Note that the portions of the file being executed as instructions are not stored in processor nor in memory registers from which they can be copied by users. Thus, the execute privilege differs from the read privilege. Also, this access privilege differs from the e (execute) access of the Bell-LaPadula model in the sense that it allows the observation of the state of the program executing a file, whereas the execute privilege of the Bell-LaPadula model does not. The execute and read combination of the Bell-LaPadula model maps directly into the execute (x) privilege of Secure Xenix.

null (-) A process with the null privilege for a file cannot access the file in any way. The Bell LaPadula model does not include the null privilege (although the execute privilege semantics comes close to it).

setuid-gid (suid-gid) Files containing program code have an additional privilege bit that can change the identity (i.e., UID or GID) of the process while executing in that file. This is discussed in the section that describes the discretionary access control in Secure Xenix.

In summary:

Bell-LaPadula privilege corresponds to File Privilege:

e (execute)	→	-
r (read)	→	r
re (read & execute)	→	x
a (append)	→	w
w (write)	→	rw
—	→	null

(2) *Privileges for Special Files, Pipes, Message Queues, Shared Memory Segments, Xenix Shared Data Segments and ACLs*

The privileges for these types of objects are the same and have the same meaning as the file privileges. They have the same relationships to the Bell-LaPadula privileges as those of files (discussed above). The only difference between the privileges for this group of objects and file privileges is that the execute privilege (x) has no meaning for this group of objects and, therefore, this field is ignored for all objects in this group.

In summary:

Bell-LaPadula privilege corresponds to this Group Privilege:

e (execute)	→	-
r (read)	→	r
re (read & execute)	→	x(ignored)
a (append)	→	w
w (write)	→	rw
—	→	null

(3) *Directory Privileges*

read (r) A process granted read access to a directory can execute instructions that cause directory attributes and contents to be fetched (read) from the directory into processor or memory registers that can be manipulated (e.g., copied) by users. Note that no information about the objects named by that directory can be retrieved. The relationship of this access to the read access of the Bell-LaPadula model is the same as that of the files.

search (x) A process granted the search privilege to a directory can execute instructions that match a given string of characters to those of a directory entry. Note that the search privilege is weaker than the read privilege, which could also be used for searching. The read privilege of the Bell-LaPadula model maps into the search privileges with the appropriate restriction; i.e., the read privilege must be restricted to directory-entry reads. Also note that the distinguished Root directory has the search privilege on for all processes in the system.

execute The execute privilege has no meaning for directories. Thus, the execute and read privilege combination if the Bell-LaPadula model has no meaning either for Secure Xenix directories. Note, however, that the execute privilege bit is reassigned by the access authorization mechanism to the search operation and thus it denotes the search permission.

add_entry (w) A process granted the add_entry (w) privilege to a directory can execute in-

delete_entry (w) structions that cause new entries to be appended to or removed from that directory. The append privilege (a) of the Bell-LaPadula model maps directly into this privilege for directories.

(rw) The Bell LaPadula write (w) access maps directly into the delete_entry privilege (rw) of Secure Xenix.

null (-) The null privilege has the same interpretation for directories as that for files.

Bell-LaPadula privileges correspond to Directory Privileges:

e (execute)	→	—
r (read)	→	r (read) or x (search ⟹ restricted read)
re (read & execute)	→	(x) no meaning
a (append)	→	w (add entry or delete entry)
w (write)	→	rw
—	→	null

(4) *Privileges for Semaphores and Xenix Semaphores*

The access privileges for System V semaphores are defined in the same way as those for files, and their relationship to the Bell-LaPadula privileges is the same as that of files. The xecute (x) privilege has no meaning for semaphores and is ignored by the access authorization mechanism. The write (w) privilege in isolation has no meaning for System V semaphores. Whenever the write privilege is on but the read privilege is off the write privilege is ignored by the access authorization mechanisms. Thus, the only non-null accesses defined for System V semaphores are read (r) and read and write (rw).

For Xenix semaphores, the execute (x) privilege has no meaning and is ignored by the access authorization mechanisms. Although the write privilege has meaning on semaphores in general, the Secure Xenix access authorization mechanism reassigns that meaning of write to the read privilege and ignores the write privilege. thus, the read (r) privilege for Xenix semaphores implies both observation and alteration and, therefore, it is equivalent to the write (w) privilege of the Bell-LaPadula model, and to read&write (rw) in Xenix.

In summary,

Bell-LaPadula Privileges correspond to System V Semaphore Privileges

e (execute)	→	—
r (read)	→	r (read)
re (read & write)	→	x (ignored)
a (append)	→	w (ignored whenever read is off)
w (write)	→	rw (read and write)
—	→	null

Bell-LaPadula Privileges correspond to Xenix Semaphore Privileges

e (execute)	→	—
r (read)	→	r (read)
a (append)	→	w (ignored)
w (write)	→	r (read and write)
—	→	null

(5) *Privileges for Processes*

The only privileges defined for processes (not to be confused with the process code file) are signal, kill, and null. The signal and kill privileges are implemented implicitly for every process and are a stylized form of a "write" to a process body. The null privilege is also implicitly implemented by the kernel through the process isolation mechanism; namely, two isolated processes have null privileges to each other.

3.1.4. The Current Access Set in Secure Xenix (B)

The current access set B is a subset of $S \times O \times A$. In Secure Xenix, the current access set is represented by a per-process data structure for some types of objects and by a per type data structure for some other types.

(1) *The Per-Process Component*

The per-process component of the current access set consists of a set of descriptors (fd) stored in the u_ofile structure of the per-process u_block. These descriptors point to a file table whose entries contain the current access privileges of the process to: files, special files, ACLs, named pipes, Xenix semaphores and Xenix shared data segments, and directories. The file-table entries are multiplexed among objects of all processes. Each per-process descriptor points to an entry in the file table. The access privileges of each entry are a subset of the privileges that the process has to the object (discussed in the next section). Note that for semaphores and for shared data segments the current-access-privilege set is the same as the process always has to these objects; i.e., the same as the corresponding access matrix entry.

(2) *The Per-Type Component*

The per-type component of the current access set consists of special descriptors that contain the access privileges available to each process. These descriptors are semid_ds for System V semaphores, msgid_ds for message queues, and shemid_ds for shared memory segments. The ipc_perm field of these descriptors contain the access privileges a process has to these objects. Here, as for Xenix semaphores and shared data segments, the current access-privilege set is the same as those the process always has to these objects.

3.1.5. The Access Matrix in Secure Xenix (M)

The access matrix M of the system state is interpreted in Secure Xenix through a set of system structures maintained by the kernel. The system structures interpreted for each object as access matrix entries are either access control lists (ACLs) or Xenix (Unix) specifications but not both. These structures represent the storage of the access matrix by columns. That is, each object is associated with a list of users that can access the object, each user having a set of access privileges restricting his access. Access control lists and Xenix (Unix) specifications are two different ways of storing the access matrix by column.

An ACL is a set of <principal identifier, access privileges> pairs that is attached to an object. The principal identifier is a non-reusable, two part identifier consisting of a user identifier and a group identifier (UID and GID). The user identifier places each individual user in a separate access control group by himself, uniquely. The group identifier places users in groups whenever such users are related by, or cooperate in, some activity or project. Such groups imply that their members have similar access privileges to a set of objects. A user may belong to several groups; however, at login time he must specify the group in which he wants to be for that login session. If no group is specified at login time, a default group is assigned to the user. Both group-membership and group-default definition on a per user basis are determined by the System Security Administrator (SSA). Default group specifications can be changed by the SSA at the user's request. Note that not all members of a group must be known when the group is formed. Members of a group may be added and deleted by the SSA subsequently.

To simplify principal identifiers, a DON'T CARE (i.e., "wild card") notation has been added [Saltzer 74]. A DON'T CARE in a user or a group field of a principal identifier is denoted by an asterisk (*). For example, the identifier Jones.Networks_FSD puts a user Jones in the Networks_FSD group. By contrast, the identifier Jones.*. names a user Jones in *any group*, whereas the identifier .*.Networks_FSD names *any user* in the Networks_FSD group. The inclusion and exclusion of individual users on ACLs and the review/revocation of privilege mechanisms are presented in [Gligor 86].

Both ACLs and Xenix protection specifications are associated in a one-to-one correspondence with the object they protect. For example, for the objects that have file system representation, the object i-node number is used to identify unambiguously its ACL. The ACL is destroyed upon object (and i-node) destruction. For objects that have file system representation the Xenix protection specifications are kept in the i-node itself. For objects that do not have file system representation (i.e., System V semaphores, message queues and shared memory segments), the ACL or the Xenix protection specification are associated with the object through the object's descriptor (i.e., semid_ds, msgid_ds, and shemid_ds). For example, the ACL's i-node number is stored in the descriptor; the Xenix specification themselves are stored directly in those descriptor and used whenever ACLs are not specified.

3.1.6. The Security Function (F)

The definition of the *security levels* as binary encodings, of assignment of print names to binary encodings, and of the (lattice) relationships between security levels is provided in [Gligor 86]. In this section we focus on the definition of the three components of the security function, namely, the assignment of maximum security level (clearance) to each subject, the current security levels (clearance) of each subject, and the assignment of security level (classification) to each object.

The assignment of user clearances in Secure Xenix is performed by the SSA on an individual and group basis in the user security profile database. The individual user clearance consists of a User Maximum Level (UML), and the group clearance consists of a Group Maximum Level (GML). These values can only be assigned and manipulated by the SSA, and must be in the range $System\_High \geq UML, GML \geq System\_Low$ for the System_High and System_Low values defined by the SSA. The *subject maximum clearance* is the greatest lower bound (viz., [Gligor 86]) of the UML and GML.

The *current subject clearance* is called the current process level (CPL), and is assigned to that process for its entire lifetime. The CPL is determined at process creation time and must be between the process maximum level (PML) and System_Low. The PML is the greatest lower bound of the UML, the GML, and the terminal maximum level (TML). (Note that, because the TML is no greater than the workstation maximum level (WML), the WML is never lower than the PML. The TML and WML are discussed below.) The CPL of a process is the user Requested_level at login time, or the user Default_level if no level is requested, if and only if the Requested_level/Default_level is less than or equal to the PML (or equivalently $\leq UML$ *and* $\leq GML$ *and* $\leq TML$). Therefore, it is clear that the subject maximum clearance always dominates the current subject clearance in Secure Xenix.

Note that a login fault is detected during the computation of the CPL (and PML). The fault occurs whenever the terminal minimum level (TmL) is greater than the user maximum level (UML) or the group maximum level (GML). Consequently, an audit record is written. The reason for this action is that the user is likely to try to login from a security area where he does not belong. Also note that a user can always request a level that is lower than both the PML and TmL, so long as both that user's GML and PML are no lower than the TmL. No login fault occurs in this case.

The assignment of *object classifications* consists of the assignment of classifications to the workstation components and the assignment of classification to the user-created objects. The assignment of classifications to workstation components is performed by the SSA (during the definition of workstation security profile), whereas the assignment of classifications to user-created objects is done by the Secure Xenix kernel. (The current level of the workstation devices is also assigned by the kernel.)

The definition of the workstation security profile is performed by the system security administrator, and includes the following classification ranges:

(i) The individual workstation classification range; i.e., workstation maximum security level (WML) and System_Low.

(ii) The classification range of each individual terminal and private devices that are connected to each workstation; i.e., terminal maximum and minimum level (TML, TmL) and the private device maximum and minimum levels (PDML, PDmL).

The assignment of these values to a specific Secure Xenix configuration is performed by the SSA and depends on the operational and the physical security environment. For example, in some operational environments the System_High and System_Low, and all other security levels, may have the same clearance value but different category sets. In such environments, the security levels assigned to individual workstations, devices and file system depend solely on the "need to know" basis.

The dependency of the security level ranges on the physical security is equally important. For example, the workstations located to areas accessible to users cleared at low security levels have a lower classification than that assigned to workstations located in areas where all users are cleared at the highest level. Physical security considerations may also require (1) that the maximum level of a terminal or private device be lower than that of its workstation (TML/PDML < WML), and (2) that the minimum level of a terminal be higher than System_Low (TmL/PDmL > SL). Terminals and other workstation devices may be located in a different physical security area than that of its workstation, and, thus, the TML/PDML may be lower than the WML. (Terminals and other private devices are also vulnerable to the additional threat of spoofing, and thus some information contained in the workstation may not be displayed on the terminal or on the private device.)

A user can only change the level of a private device or of a terminal to a level that he requests at login time (viz., the computation of the CPL). The current level of a private device or of a terminal can be displayed by the kernel on request. The minimum level of a terminal or of a private device classification may be higher than System_Low because physical security considerations may require that individuals with a low clearance, or with no need to know, may be denied access to workstations, terminals and private devices located in highly classified areas or in areas with different "need to know". This is done by raising the TmL/PDmL to a correspondingly high security level.

A workstation terminal, or a private device, also has a current classification, called the Current Terminal Level, or the Current Private Device Level (CTL or CPDL). In Secure Xenix, both the CTL and CPDL equal the CPL of the user, system process, or daemon to which they are attached (and that owns or opens them). Note that it is possible to have CTL < TmL because CTL = CPL and CPL equals Requested_Level < TmL of a user whose UML, GML $\geq$ TmL. For similar reasons, it is possible to have CPDL < PDmL.

The determination of the classifications of the user-created (or opened) objects is performed by the Secure Xenix kernel, and consists of the following three groups of rules.

(1) Classification of Files, Special files,
Xenix Semaphores, Xenix Data Segments, and ACLs

Objects in this group have a single level for their entire lifetime. (Exception to this are the special files whose activation level equals the level of the process that activates or opens them. None of the Xenix special files retain any state information in current configuration. Whenever such files retain state information, SSA intervention is required for activation.) That is, unless a special trusted process with discretionary access to that object changes the object classification (i.e., downgrade or upgrade), the object classification does not change. The classification of an object in this group is the CPL of the creating process and must be equal to the security level of the directory containing that object.

An object in this group can only be destroyed by a process with the same (CPL) level as that of the object; the object is destroyed only if its reference count equals zero (i.e., it is not shared by any other directory or process). Note that special files are not destroyed; they are only closed.

(2) Directory Classification

A directory has a single security level for its entire lifetime, just as in the case or ordinary files. However, unlike ordinary files, the security level of a newly-created directory can be assigned from a range of levels. The lowest level of the range is the CPL of the creating process and must be equal to that of the directory that contains the newly-created directory. The highest level of the range is the WML. If a process creates a new directory but does not request any level for that directory, the default level of the directory is that of the process (i.e., the CPL) and that of the containing directory. The classification of a directory does not change during the lifetime of the directory unless a trusted process with discretionary access to that directory always changes it.

A directory can only be destroyed by a process at the same level (i.e., CPL) as that of the containing (parent) directory. Also, a directory can only be destroyed if it contains no files. This Xenix interface convention introduces a covert channel, which is discussed in [Gligor86] because a lower level process can discover whether a higher level process has removed all the files from the higher level directory when it tries to remove them.

(3) Classification of Processes, System V Semaphores, Message Queues and Shared Memory Segments

The security levels that are assigned to these objects by the classification rules of the kernel always equal the CPL of the process that created these objects. Similarly, these objects can only be destroyed by the process that created them or by a trusted process at the same level as that of the objects. The classification of those objects does not change during their lifetime unless a trusted process with discretionary access to those objects changes it.

3.1.7. Hierarchy (H)

The only Secure Xenix objects that may contain multiple components with different classifications are directories. Thus, the only object hierarchy in the system for the objects that have a file system representation is that provided by the directory hierarchy. All objects in this group (i.e., group (1) above) are classified at the level of the creating process, which must equal that of the directory containing the object.

Objects that do not have file system representation (i.e., objects in group (3) above) are classified at the level of their creator process. This ensures that these objects cannot be at a lower level than that of the processes' current directory. This also maintains the "nondecreasing level" rule for the directory hierarchy. These objects form the isolated points (i.e., the "stumps" in the Bell-LaPadula terminology [Bell76]) of the hierarchy.

The rules for assigning specific classifications to directories in the hierarchy prevent a process from placing a newly-created directory in another directory at a lower level than that process' CPL. However, a process can create an "upgraded" directory that has a higher level than that of the CPL of the creating process and that of its containing directory.

Note that a user process can create links in its current directory to objects that have file system representation. However, links to directories can only be created by trusted processes. User processes can only link (non-directory) objects in the process current directory (i.e., CPL=directory level), and only if the security level of the object being linked equals that of the current directory.

The Secure Xenix hierarchy has a ROOT directory whose level is always System_low. All processes have the search privilege (x) to this directory.

3.2. State Transitions in Secure Xenix

Transitions from state to state are defined by the kernel calls and returns of Secure Xenix. Thus, each rule ρ_i in

$$\rho: R \times V \rightarrow D \times V,$$

of the Bell-LaPadula model is represented as follows:

(1) Each request $R_k \in R$ is represented by a specific kernel call or by a trusted process call (these calls are implemented by kernel calls). R is the set of all kernel and trusted process calls.

(2) Each input to R_k comes from the current system state V. That is, both parameters explicitly passed to each call

(such as object identifiers, values, pointers, access privileges, and so on) and parameters implicitly passed to each call (such as the system hierarchy, security levels, and so on) belong to the current system state.

(3) Each decision $D_m \in D = \{\text{Yes, No, ?, Error}\}$ is represented by a specific return to a kernel call. "Yes" is represented by the successful return parameter. "No" is represented by the error return parameter that corresponds to violations of the access control (e.g., mandatory or discretionary checks). "?" is represented by the error return parameters specifying that the kernel call parameters are faulty (e.g., non-existent file, parameters out of range, attempt to invoke a privileged kernel call, etc.). In general, these error returns are called domain errors. "Error" is represented by error returns that correspond to other exceptional conditions detected during the execution of specific kernel calls (e.g., deletion attempted on a non-empty directory, overflow conditions, etc.). Note that all decisions represent some information from the system state at the time of the kernel call, V, or from the new system state, V*, entered by the system as a consequence of the call.

(4) Whenever $D_m \neq$ No, $D_m \neq$?, and $D_m \neq$ Error, the output of R_k includes a new new state V*, in addition to D_m=Yes. The new system state may include new objects, a new hierarchy, or may exclude some objects and access privileges from previous states, and so on.

The D_m's, the characteristics of the expected and of the new state for each R_k are described in the Secure Xenix DTLSs.

3.3. Access Control in Secure Xenix

In this section we report the invariant access control checks that are performed in Secure Xenix. This includes the presentation of (1) authorization checks for mandatory control, (2) authorization checks for discretionary access control, including the Setuid-Setgid mechanism, and (3) the computation of the effective access authorization to objects.

3.3.1. Mandatory Access Authorization

The authorization rules are divided into three groups depending on the type of object being accessed.

(1) The object is a File, a Special File (Device), a Directory or a Shared Memory Segment or an ACL:

A process may Read (Execute) an object if the CPL of the process is GREATER THAN or EQUAL TO the classification of the object.

A process may Write an object if the CPL of the process is EQUAL TO the classification of the object.

This rule implies that the data displayed on a private device or on a terminal can be a level that is no higher than that of the CPL and, implicitly, of the CPDL/CTL. Any displayed data that may have to be at a lower level can be labeled separately by a trusted process of the secure application itself. Thus, the application is responsible for providing labels for data fields that would be appropriate for, and that use, the different terminal features (i.e., windowing, scrolling, etc.).

(2) The object is a Named Pipe, Semaphore, Message Queue, Xenix Shared Data Segment:

A process may Read/Write (open/close) an object if the CPL of the process equals the classification of the object.

(3) The object is a Process:

A process can signal (kill) another process if the CPL of the latter is GREATER THAN or EQUAL TO that of the former.

(4) For all objects, a process has NULL access to an object if the CPL of the process is ISOLATED FROM the classification of the object.

The above rules imply that the flow of information in Secure Xenix can only take place from a given level to another level that is no lower than the first.

The mandatory access authorization rules presented above are invariant for all Secure Xenix kernel calls. That is, depending on whether a kernel call is relevant to a particular type of object, one or several of the above rules apply to that call. These rules are compatible with the ss-property and the *-property of the Bell-LaPadula model for the following reasons.

(1) Rules 1 and 2 of Secure Xenix imply conditions (ii) and (iii) of the *-property.

(2) Rule 3 of Secure Xenix *implies* condition (i) of the *-property.

(3) Since the subject maximum clearance (i.e., the greatest lower bound of UML and GML) always dominates the current subject clearance (i.e., CPL) in Secure Xenix, Rules 1-3 above *imply* the ss-property.

However, it should be noted that *equivalence* between the ss-property, the *-property of the Bell-LaPadula model and any system interpretation is impossible in practice. There are two reasons for this.

First, consider the meaning of the execute (*e*) privilege of the Bell-LaPadula model presented in section 2.4 above. This privilege does not exist in practice because, in any system, the execute privilege implies some observation of the behavior of the object being executed. Therefore, in practice, the execute (*e*) privilege must be eliminated from condition (i) of the ss-property and added to condition (ii). Furthermore, it must be also added to condition (iii) of the *-property; otherwise, observation of objects at levels that are higher than those allowed to the user or his process is possible.

Second, consider the implementation of the "append" operation that requires the append privilege (*a*) of the Bell-LaPadula model. In practice, append operations may require one or more of the following observations of objects or system state:

(1) find the end of the object that is the target of the append operation;

(2) find the name of an object in a directory at a higher level than that of the process executing the append operation;

(3) find out whether the object exists;

(4) find out whether the append operation fails due to a storage channel exception.

Consequently, in practice, the append operation implies not only alteration of an object but also observation of the object or of the system state. Therefore, in practice, the append (a) privilege must be eliminated from condition (i) of the ss-property and added to condition (ii) of the *-property for the similar reasons to those mentioned for execute (e) above.

With the above two modifications that are required in practice, the ss-property and the *-property would be *equivalent* to the rules 1-3 of the Secure Xenix implementation. Note, however, that consistency of the Secure Xenix interpretation with the model only requires that Rules 1-3 above *imply* the ss-property and the *-property.

3.3.2. Discretionary Access Control

The discretionary access authorization rules of Secure Xenix define the Secure Xenix model of discretionary policy. Discretionary policy is characterized by four classes of axioms, namely, (1) authorization axioms, (2) axioms for distribution of access privileges, (3) axioms for review of access privileges and (4) axioms for the revocation of access privileges. The informal specification of the first three classes of axioms are required explicitly by the [TCSEC 83] in the discretionary access control area of B2-class system. The informal specification of the fourth is required implicitly in the statement that "the enforcement mechanism shall allow users to specify and control sharing for these objects.

(1) Authorization in Secure Xenix

The specification of the discretionary authorization mechanisms of any system consists of two parts. First, it must include a specification that relates every (kernel) operation on one or more objects with the privileges required by the operation for those objects. This is necessary because the authorization mechanism requires different (combinations of) privileges for different operations. Lack of such specification could mean that the wrong privilege may authorize an operation. As seen in section 3.1.3 above, the correspondence between an access privilege to a kernel operation depends on the type of objects and is not entirely obvious.

Second, the discretionary authorization mechanisms must include a specification of how the current access privileges of subjects are related to the specification of the subjects access to objects by the access matrix. This relationship is defined by the ds-property of the Bell-LaPadula model, and is important because it relates the high-level, human-oriented, discretionary access controls specified by the access matrix with low-level, human-oriented, discretionary access controls specified by the access matrix with the low-level, system-oriented, discretionary controls of the system.

It should be noted that the requests R_k discussed below, namely, *CALL, REVOKE, REVIEW, ACCESS, GRANT, EXCLUDE* are implemented by kernel calls or sequences of kernel calls that require the reading and writing the ACL and Xenix specifications. *ACCESS and CALL* are implemented by a single kernel call (i.e., "access" and "exec").

The two general requirements of discretionary authorization can be expressed by the following two axioms. Let $\rho : R \times V \longrightarrow D \times V^*$ be the set of rules.

For all R_k executed by S_i on some objects O_j with $R_k \neq$ CALL, GRANT, REVOKE, REVIEW, ACCESS, EXCLUDE, and $\underline{x}$ are the required access privileges for R_k

(1.1) $(D_m = \text{YES}) \Longrightarrow (S_i, O_j, \underline{x}) \in B$, and

(1.2) $(S_i,\ O_j,\ \underline{x}) \in B \Longrightarrow \underline{x} \in M_{ij}$ [Bell-LaPadula 76].

Secure Xenix satisfies both requirements mentioned above. First, the DTLSs of Secure Xenix specify the discretionary privileges for each type of object that are required by each kernel call. Furthermore, the kernel call fail whenever the required privileges are not among the privileges of each object used by the call. Second, each current access of a process to an object is derived from either the objects' ACL or from its Xenix specifications (i.e., i-node, semid-ds, msgid-ds, shemid-ds) when the object is open or created; viz., section 3.1.4 above. Because these data structures represent the access matrix in Secure Xenix (viz., section 3.1.5 above), the ds-property of the Bell-LaPadula model is also satisfied.

(2) Distribution of Access Privileges in Secure Xenix

The policy for the distribution of access privileges must specify how "the access permission to an object by users not already possessing access permission shall only be assigned by authorized users" [TCSEC 83].

In Secure Xenix, the only users that are authorized to distribute object privileges to other users are the *owner* of those objects. Ownership of an object is a user attribute and not an access privilege. In Xenix, ownership is determined solely by the user identifier (and not by the group identifier GID). Each object has only one owner and only the owner is authorized to modify either the ACL or the Xenix specifications for his objects.

This privilege distribution policy is succinctly stated by the following two axioms:

(2.1) Ownership Axioms:

$$\forall i \neq j,\ \forall S_i, S_j \in S',\ S_i = Owner(O_i) \Longrightarrow S_j \neq Owner(O_i),\ and$$
$$S_i = Owner(O_i) \Longrightarrow \forall \underline{x} \in A, \underline{x} \in M_{ii}$$

(2.2) Privilege Granting Axiom:
Let $\rho : R \times V \longrightarrow D \times V^*$ be the set of rules.
For all R_k executed by S_i on objects O_j with $R_k = GRANT(\underline{x},\ S_p)$

$(D_m = \text{YES}) \Longrightarrow [S_i = Owner(O_j)]$ and $\underline{x} \in M_{pj}$

The effects of the privilege granting are equivalent to the *inclusion* of a user/group identifier on an ACL or in the Xenix specifications. The inclusion of users on ACL's is explained in section 3.1.5 above and on Xenix specifications for an object in [Ritchie 74].

Similarly, the policy for the distribution of access privileges must be able "to specify a list of named individuals and a list of groups of named individuals for which no access to the object is to be given".

In Secure Xenix this is possible since the owner can either decide *not* to include a specific user or group in the ACL or Xenix access specification or to *exclude* a specific user's or group's access as explained in section 3.1.5 above.

(2.3) Privilege Exclusion Axiom:
Let $\rho : R \times V \longrightarrow D \times V^*$ be the set of rules. For all R_k executed by S_i on $M[S_i, O_i]$ with $R_k = EXCLUDE(\{S_j\},\ O_i)$
$(D_m = \text{YES}) \Longrightarrow [\forall j \neq i,\ S_i = Owner(O_i)$ and $(S_j, O_i, \phi) \in B$, where $\{S_j\} \subset S']$.

(3) Review of Access Privileges in Secure Xenix

The policy for review of access privileges "shall be capable of specifying, for each named object, a list of named individuals and a list of groups of named individuals with their respective models of access to that object" [TCSEC 83].

In Secure Xenix, the only users that can perform access review (i.e., reading the ACL or the Xenix specifications) for an object are the owners of that object. However, any user can inquire whether he has access to an object, and what type of access, regardless of the object's ownership.

This can be succintly stated by the following axioms. Let $\rho : R \times V \longrightarrow D \times V^*$ be the set of rules.

(3.1) For all R_k executed by S_i on $M[S_i, O_j]$ with R_k = REVIEW(O_j):
$(D_m = \text{YES}) \implies S_i = Owner(O_j)$

(3.2) For all R_k executed by S_i on O_j with R_k = ACCESS(O_j):
$(D_m = \text{YES}) \implies S_i \in S'$.

(4) Revocation of Privileges in Secure Xenix

The policy for revocation of privilege must specify how access privileges for an object can be taken away from users that have these privileges in a selective manner and, possibly, partially.

In Secure Xenix the (selective and partial) revocation of access privilege can be performed only by the owner of an object. The reason is that only the owner of the object may modify ACLs and Xenix specifications. This can be expressed succinctly by the following axiom. Let $\rho : R \times V \longrightarrow D \times V^*$ be the set of rules.

(4.1) For all R_k executed by S_i on $M[S_i, O_j]$ with R_k = REVOKE($\underline{x}, S_p$) :
$(D_m = \text{YES}) \implies [\forall i \neq p,\ \underline{x} \notin M_{pj}$
$and\ S_i = Owner(O_j)]$

(5) The Setuid-Setgid Mechanism of Secure Xenix

The SETUID protection mode is used to build controlled interfaces to various objects [Ritchie 74]. Whenever a program with the SETUID bit is executed, the invoking process inherits the privileges of the program owner. Every process has both a real and an effective user identifier that are identical except when a SETUID program is executed. Then the effective user identifier is set to that of the program owner. All discretionary access control decisions are based on the effective user identifier and not on the real one. (There is a similar mechanism called the SETGID mechanism for changing the effective group identifier.)

Although the SETUID feature can be very useful it also poses three types of security risks: first, a poorly designed SETUID program can compromise the program owner's security; second, the code of the SETUID program may be modified in an unauthorized way; third, a Trojan Horse in a borrowed program may steal a user's privileges by creating a SETUID program.

The modifications to the SETUID/GID mechanism presented in [Gligor 86] make it impossible for a user to change an existing SETUID program, or for a Trojan Horse to steal user's privileges by creating a SETUID program. However, it is the responsibility of the user to use extreme care in the design of SETUID programs. The operating system cannot protect the user from his own mistakes. Even if the user is careless or malicious, he can only hurt himself by misusing the modified SETUID features mentioned above because he cannot create a SETUID program under a different user's identifier. Note that the mandatory access control (discussed below) remains unaffected by the SETUID mechanism.

The SETUID/GID mechanism of Secure Xenix enforces the *separation of privileges* between the subject that invokes a SETUID/GID program and the subject that owns the program. This means that a subject invoking a SETUID/GID program may only have *indirect* access to some of the objects of the SETUID/GID program owner. This can be expressed succinctly by the following axiom:

(5.1) Let $A = \{r,\ w,\ x,\ null,\ suid-gid\}$
$[S_j, O_j, indirect(\underline{x})] \in B \implies$
$CALL(S_j, O_k),\ S_i = Owner(O_k),$
$suid\_gid \in M_{ik}$ and $(S_i, O_i, \underline{x}) \in B$

In other words, if a program has the SETUID-GID bit on, the subject S_i executing it has privileges of the owner to objects O_i that may not be directly available to that subject's callers (i.e., S_j).

3.3.3. Computation of the Effective Access in Secure Xenix

The effective current access of a subject to an object in Secure Xenix follows two rules. These rules are compatible with the Bell-LaPadula model. They are:

(1) A user process is allowed to access an object in a given mode (i.e., requiring a certain privilege) only if both mandatory and discretionary checks are passed.

(2) The Error value returned for failed discretionary checks must be the same as that returned from failed mandatory checks unless the mandatory checks have passed.

The first rule is necessary because, otherwise, the requirement of the secure states and the Basic Security Theorem of the Bell-LaPadula model would be violated. The second rule is necessary because otherwise leakage of information from higher levels to lower levels may be possible. That is, whenever the discretionary checks are done first, a higher level subject may revoke or add a privilege for an object to a lower level subject. The lower level subject would then distinguish between denied discretionary access and denied mandatory access errors. Thus, by modulating the discretionary access of the lower level subject to a higher level object, a higher level subject could transfer information to a lower level object. In Secure Xenix, the mandatory access checks be performed before the discretionary checks for every kernel call accessible to a user process. However, this is a stronger requirement than the more general one specified in (2) above.

3.4. Initial State (z_0)

The initial state of any Secure Xenix installation is set by a secure initialization procedure. The secure initialization consists of three distinct phases: (1) system configuration and generation, (2) system and user security profile definition, (3) normal startup (or Initial Program Load - IPL). The first phase is performed by the TSP (Trusted Systems Programmer) in maintenance mode. Once this mode is left, the TSP functions are automatically disabled, and only the rest of the administrative users have access to the workstation. The second phase work is performed by the SSA. The third phase work, normally the IPL, is performed by anybody with physical access to the power on/off switch of the workstation.

The IPL of the Secure Xenix can only take place with input from the fixed disk, whereas in maintenance mode, the IPL can only take place with input from the removable media (e.g., diskette) drive. This separation of IPL input is enforced by a special hardware configuration that, when installed by the TSP, prevents user mode IPL from using the removable media drive. No cryptographic authentication of the removable medium [Gligor 79] is performed at this time. The TSP is the only administrative user that has access to the internal hardware configuration, and he would have to be trusted to configure the system correctly anyway. (If the TSP is not trusted, on-site physical surveillance methods would become necessary, cryptographic authentication notwithstanding).

During the Secure Xenix IPL, several consistency checks are performed. Xenix already performs file system consistency checks (i.e., through the "fsck" program). In particular, the IPL recovers whenever the system is started up after an improper shut-down (i.e., after a crash, after power-off during disk I/Os, etc.) This ensures that security label consistency is maintained because each label is written onto the disk with a separate, atomic sector-write operation. In addition to the file system consistency checks, Secure Xenix checks (1) the consistency of the security map, (2) the consistency of the current obje-t label, and (3) the consistency of the overall security level hierarchy (i.e., the non-decreasing security levels for directories). This is done by the "scheck" program.

3.5. Compatibility in Secure Xenix

The interpretation of the compatibility axiom in Secure Xenix requires that a directory contains (1) non-directory objects (which have file system representation) only at the same level as that of the directory, and (2) directory objects at the same level as that of the directory or higher. Consequently, if a directory is at a higher security level than that of a subject, all objects filed in that directory remain inaccessible to the subject.

The rules for object classification discussed in section 3.1.6 above, and the definition of the Secure Xenix hierarchy discussed in section 3.1.7 above, imply that the compatibility axiom is satisfied.

3.6. Tranquility in Secure Xenix

The section 3.1.6 is specified that both the current process level (clearance) and the classification of objects in Secure Xenix do not change during the lifetime process and of an object, respectively (unless a trusted process with discretionary access to those objects or with root privileges changes those levels). This suggests that the kernel call accesses, and the clearance and classification rules of Secure Xenix satisfy the tranquility principle of the Bell-LaPadula model.

3.7. Activation

The design of Secure Xenix satisfies the two activation axioms defined in section 2.4.6 above. First, an active object can become inactive only through destruction. Objects that have file system representation are inactivated by the destruction of their i-nodes and by the deallocation of the appropriate table entries (i.e., file descriptor, file table entry). Objects that do not have file system representation are inactivated by the destruction of their descriptors and of their table entries (i.e., shared memory, semaphore and message queue table entries). A process is destroyed by destroying the corresponding process table entry. Consequently, the destruction of all these objects makes them inaccessible to any active process.

Second, whenever an object is created (activated) the state of the object is automatically written by the kernel with zeros. Thus, the previous states of this object, or of any other destroyed object whose storage representation is being reused, are erased before reuse. This is discussed in more detail in a separate document on object reuse. Thus, the state of a newly activated object cannot depend on the state of *any* previous object incarnation.

Note that the destruction (inactivation) of some objects, such as some special files representing terminals, do not cause the object representation to be "erased." Whenever such objects do not retain state information they can be reactivated and made accessible to different processes (and labeled accordingly; viz. section 3.1.6 above). However, the activation of objects that retain state information after their deactivation requires the intervention of the SSA and of trusted processes (i.e., mount/unmount volumes).

Secure Xenix also satisfies an additional activation axiom that define the classification of a newly activated (created) object and the object destruction rule.

Consistency with the compatibility axiom and with the *-property requires:

(3) *Classification of Newly Activated Objects and the Object Destruction Rule*

Let $O = O' \cup O''$, where $O'(O'')$ = active (inactive) objects, and

new(o) = $\{O' := O'' - o$ and $O' := O' + o\}$, and

destroy(o) = $\{O' := O' - o$ and $O'' := O'' + o\}$

$\{$CALL$[Si$, new(o)$]$ or CALL$[S_i$, destroy(o)$]\} \Longrightarrow$

$\{H^{-1}(o) \neq \phi => f_o(o) \geq f_c(S_i) = f_o[H^{-1}(o)]$ or $H^{-1}(o) = \phi => f_o(o) = f_c(S_i)\}$

where H^{-1}(o) is the parent of object o.

The interpretation of these axioms in Secure Xenix is discussed in section 3.1.6 above.

4. Conclusion

In this paper we have reviewed the Bell-LaPadula model for secure systems in its most complete form. We also defined the interpretation of this model in Secure Xenix has been defined. We showed that the access control mechanisms of Secure Xenix satisfy the axioms of the Bell-LaPadula model.

5. References

[Bell 73] Bell D.E., and L.J. LaPadula, "Secure Computer Systems," Air Force Elec. Syst. Div. Report ESD-TR-73-278, Vols. I, II, and III, November, 1973.

[Bell 74] Bell D.E., and L.J. LaPadula, "Secure Computer System: Mathematical Foundations and Model," MITRE Corp., Bedford, MA, (September 1974).

[Bell 76] Bell, D.E. and LaPadula, L.J., "Secure Computer System: Unified Exposition and Multics Interpretation," MITRE Corp., MTR-2997, 1976 (available as NTIS AD-A023588).

[Feiertag 77] Feiertag, R.J., Levitt, K.N. and Robinson, L., "Proving Multilevel Security of a System Design," Proc. of the 6th ACM Symp. on Op. Syst. Princ., Lafayette, Ind., 1977, pp. 57-65.

[Gligor 79] Gligor, V.D. and Lindsay, B.G., "Object Migration and Authentication," *IEEE Trans. on Software Engineering*, SE-5, no. 6, (Nov. 1979).

[Gligor 86] Gligor, V.D., Burch, E.L., Chandersekaran, C.S., Chapman, S., Dotterer, L., Hecht, M.S., Jiang, W.-D., Luckenbaugh, G.L., and Vasudevan, N., "On the Design and Implementation of Secure Xenix Workstations," Proc. of IEEE Symp. on Security and Privacy, Oakland, Calif., April 1986.

[Ritchie 74] Ritchie, D.M., and Thompson, K., "The Unix Time-Sharing System," *Comm. of the ACM*, Vol. 17, No. 7, (July 1974), p.p. 365-375.

[Saltzer 74] Saltzer, J.H., "The Protection and Control of Information Sharing in Multics," *Comm. of the ACM*, Vol. 17, No. 7, (July 1974), p.p. 388-402.

[TCSEC 83] Department of Defense - Computer Security Center, "Trusted Computer Systems Evaluation Criteria," Final Draft, August 1983.

INFORMAL VERIFICATION ANALYSIS

By Barry C. Stauffer and Roger U. Fujii

Logicon, Inc.

The concern for the security of computer systems has been intensified by the increasing dependence of the system on the computers and the ability for the systems to react autonomously. The development of secure systems has proven to be an engineering challenge. While much emphasis has been devoted to perfection of formal specification techniques, to date these techniques have had only limited use in fielding state-of-the-art systems. The need for assurance of secure systems remains.

This paper presents an adaption of existing software verification and validation technology to be applied to the specific needs of computer security.

1. Introduction

Computer security includes all measures to protect against unauthorized (accidental or intentional) disclosure, modification, or destruction of computer systems, processes, and data. Also included are those measures to protect against denial of service. Of concern is the protection of classified data, mission critical data and processes, unclassified data requiring special protection (official use only data, personnel data, etc.) and integrity of data. Mission Critical Computer Systems (MCCS) have the additional concerns of process security and process integrity, i.e., to provide assurance that one process cannot inadvertently access, initiate or deny access to another critical process.

We have gained an increased understanding of security-specific technical issues from the computer security work undertaken over the last decade. Two significant developmental factors affect the security of computer systems. The first is the rigorous use of sound modern software engineering principles combined with systematic detailed program reviews. The second factor is evaluating and incorporating critical security issues during all phases of the development cycle (i.e, build computer security into the system rather than add it as a separate part).

The use of a security model is the most effective way to evaluate critical security issues. As part of the system requirements, a system security model defines the system-enforced security rules.

It specifies the access controls on the use of information and how information will be allowed to flow in the system. The model also provides the mechanism for specifying how to change access controls and interfaces dynamically without compromising system security. A precisely tailored security model can ensure that a system will contain a level of security appropriate to its intended application. Thus the security model, which defines system security needs, is a major component of a secure system.

The major remaining factor in the development of secure computer systems is the ability to validate secure system behavior. Complete and total trust in the security of system can only be achieved with a formal, mathematically sound validation that the system, as built, correctly implements its formally specified security model. The theory of formal program validation is now better understood, except for problems of concurrency and asynchronism, and a significant amount of progress has been made toward verifying the correctness of computer programs with complete mathematical integrity. At the present time, this technology can only be used to validate small programs. The time and cost to validate a large complete program, such as an operating system, precludes the use of this formal validation technology. There is, however, a rigorous, though less mathematically formal, technology that can be used to validate system security.

2. Independent Verification and Validation

Independent Verification and Validation (IV&V) is the systematic analysis, test and evaluation of a computer system by a contractor or agency independent of the developer. It is a highly structured, rigorous system engineering discipline consisting of a series of specific activities which, in the ideal, parallel program development. Its goal is to provide complete and total assurance that the delivered operational system satisfies all of its requirements and is limited to performing only its intended functions.

IEEE-STD-729 defines Verification and Validation as:

> "The process of determining whether or not the products of a given phase of the software development cycle fulfill the requirements established during the previous phase," and "The process of evaluating software at the end of the software development process to ensure compliance with software requirements."

Figure 1 is a graphic depiction of the IV&V process.

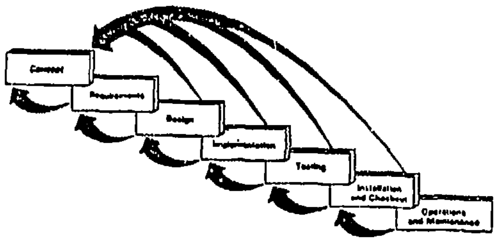


Figure 1. Independent Verification and Validation Process

IV&V was developed in the 1960s for several military and space programs with a clear need to ensure the reliability of critical software. Since its inception, the IV&V methodology has been expanded to include the analysis of the entire system. IV&V has been performed on varied MCCS including missile control, launch, guidance and maintenance software; avionics software; missile mission planning, weapons control, and flight software; satellite ground and flight systems; C3 intelligence systems.

The starting point for IV&V methodology is a criticality analysis which focuses IV&V resources on those software functions which are deemed system critical. This effort is independent of specific IV&V tasking and is used to define the nature and scope of IV&V for specific systems. Criticality analysis of nuclear missile systems, for example, has resulted in an IV&V technology known as Nuclear Safety Cross Check Analysis (NSCCA). NSCCA focuses on system critical nuclear safety issues, including prevention of unauthorized or inadvertent arming, enabling, launching, firing, or releasing of the weapon system; prevention of a faulty launch; and premature or unsafe operation of the weapon system, among other possibilities. This technology has been formalized in the Air Force AFR-122 regulations and is also being adapted in a Navy standard for Software Nuclear Safety (MIL-STD-SNS).

3. Computer Security Evaluation

IV&V and NSCCA technology have been adapted to meet the specific needs of computer security. The analysis, called Computer Security Cross Check Analysis (CSCCA) focuses on the critical computer security issues. CSCCA starts with the critical security objectives analysis to focus the effort on the security critical functions of a specific system. For example, on an intelligence collection and dissemination system these objectives would verify the developed software does not permit:

- unauthorized or inadvertent access of processes, fixing algorithms, sources, or gathered data
- deliberate or inadvertent denial of system services
- unauthorized use of system services
- deliberate or inadvertent misuse of system services

Computer security cross check analysis contains a series of activities to analyze and test the products of the development process. These activities are designed to detect as early as possible those development problems which affect security issues and to provide the program manager with increased visibility into the security requirements of the system under development. These activities include:

- System security requirements analysis
- Design analysis
- Code analysis
- Independent security testing
- System validation and recommendation for certification

The process begins with a thorough analysis and evaluation of the security requirements for the system. The purpose of this analysis is two-fold. First, to independently derive from the security instructions those security requirements that apply to the system. Second, to tailor the analysis approach to the specific needs of the system under evaluation. For maximum benefit, the analysis and evaluation of the system security requirements should be performed early in the program development cycle so that potentially conflicting or inconsistent system requirements are detected and corrected be[illegible]re the requirements are translated into program design code. Figure 2 demonstrates this analysis.

The objective of requirements analysis is to ensure that the system functional requirements, as embodied in the requirements documentation, are consistent with the system security requirements. Requirements analysis detects errors and deficiencies in the requirements which could result in a subsequent system failure to meet the critical security objectives. The analysis, Figure 3, evaluates the program requirements, interface requirements, access requirements, control flow requirements, timing and sizing requirements, etc., for compliance with the

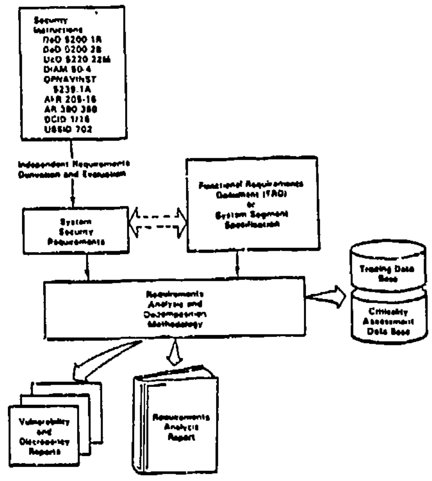

Security Instructions
DoD 5200 1R
DoD 5200 28
DoD 5220 22M
DIAM 50-4
OPNAVINST 5239.1A
AFR 205-16
AR 380 380
DCID 1/16
USSID 702
Independent Requirements Derivation and Evaluation
System Security Requirements
Functional Requirements Document (FRD) or System Segment Specification
Requirements Analysis and Decomposition Methodology
Tracing Data Base
Criticality Assessment Data Base
Vulnerability and Discrepancy Reports
Requirements Analysis Report

Figure 2. Security Requirements Independent Derivation

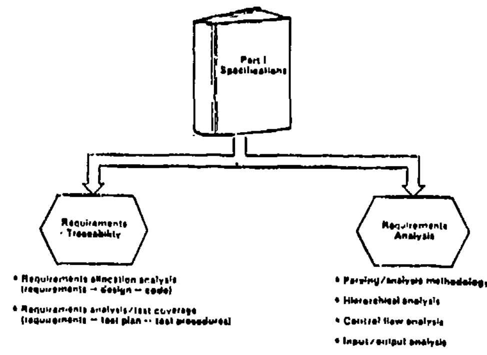

Part I Specifications
Requirements Traceability
Requirements Analysis
• Requirements allocation analysis (requirements → design → code)
• Requirements analysis/test coverage (requirements → test plan → test procedures)
• Parsing/analysis methodology
• Hierarchical analysis
• Control flow analysis
• Input/output analysis

Figure 3. Requirements Analysis Activities

security model. Requirements analysis also ensures that system requirements are properly implemented in the security kernel, if appropriate.

The objective of design analysis is to verify the system design is consistent with the system security requirements, i.e., the requirements are traced into the design. Design analysis, Figure 4, evaluates the system design for compliance with the security model and detects errors and deficiencies in the design. Design analysis evaluates the data flow, process interfaces, and overall control logic of the design in terms of its ability to implement the security requirements. The design of access controls and information flows are evaluated for correctness and consistency with security requirements. Design analysis evaluates the description, security level, and intended usage of each data item in the program design to verify that the structure, security level, and intended usage of program data will satisfy the security requirements.

Design data analysis also evaluates the relationships between secure processes and ensures that those relationships satisfy the security requirements. The design of interfaces between program components, firmware, and hardware is analyzed making certain that these interfaces have been correctly defined and do not violate security requirements. The design is correlated to the system requirements to make certain the design is a correct and complete implementation of the system requirements.

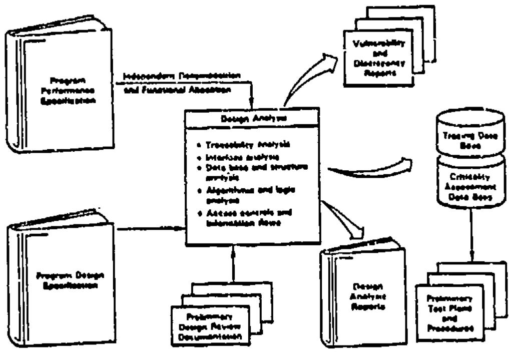

Program Performance Specification
Independent Documentation and Functional Allocation
Vulnerability and Discrepancy Reports
Design Analysis
• Traceability analysis
• Interface analysis
• Data base and structure analysis
• Algorithms and logic analysis
• Access controls and information flows
Tracing Data Base
Criticality Assessment Data Base
Program Design Specification
Preliminary Design Review Documentation
Design Analysis Reports
Preliminary Test Plans and Procedures

Figure 4. Design Analysis Activities

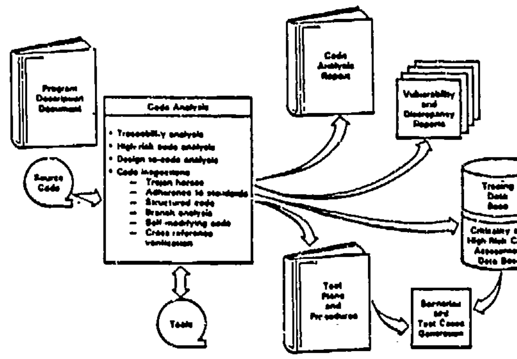

Program Description Document
Code Analysis
• Traceability analysis
• High-risk code analysis
• Design-to-code analysis
• Code inspections
– Trojan horses
– Adherence to standards
– Structured code
– Branch analysis
– Self-modifying code
– Cross reference verification
Source Code
Tools
Code Analysis Report
Vulnerability and Discrepancy Reports
Tracing Data Base
Criticality and High-Risk Code Assessment Data Base
Test Plans and Procedures
Scenarios and Test Cases Generation

Figure 5. Code Analysis Activities

Kernel analysis evaluation is a critical part of the design analysis since a kernel mediates all accesses to system resources and therefore, implements the basic security rules for the system. Design analysis must include evaluation of the correctness and completeness of the mediation process with respect to the security policy requirements. The kernel design must be carefully examined to determine that a tamper-proof implementation is feasible. For verification purposes, the kernel functions must be kept to a minimum. The design must be evaluated to determine that only those functions essential for the most critical security rules are expressed in the kernel. While it is most desirable to minimize kernel functions, some designs may include trusted processes within the security kernel. Trusted processes may be permitted to bypass certain security rules, for example, allowing downgrading for a specific application when normally the write permission would be forbidden. Trusted processes must be carefully evaluated for correctness to ensure that security will not be compromised.

Maximum benefit is derived from design analysis when it is conducted prior to coding. Design errors not detected until after coding has commenced may require redesign and recoding of significant program segments and thereby increase program cost and delay program schedules.

The objective of code analysis is to ensure that the coded program correctly and completely implements the system security requirements. Code analysis detects errors and deficiencies in the program. The analysis, Figure 5, evaluates the data descriptions, interfaces, equations, algorithms, and overall control logic of the program in terms of their adherence to the security requirements. Particular attention is paid to an analysis of the security kernel to guarantee that security-critical design functions have been properly implemented in the code. Code analysis also identifies extraneous code whose purpose cannot be traced back to the design or requirements:

The techniques used in code analysis are similar to the techniques used in design analysis. They include program logic analysis, program data analysis, and program interface analysis. For example, program data analysis examines the source data structures in conjunction with the program logic to determine whether any possible security errors such as data crosstalk or spillage, inconsistent use of data types, or improper protection of classified data are present in the source code.

The goals of CSCCA testing differ from those of the testing performed by the development organization, who tends to focus on ensuring the program is functionally correct. CSCCA testing, on the other hand, focuses on locating potential security weaknesses and identifying extreme or unexpected situations that could cause noncompliance with the critical security objectives. The security testing, Figure 6, is an independent dynamic set of tests designed to verify the results of earlier analysis, to investigate program behavior, to identify program shortcomings, and to ensure that the program complies with its security requirements. CSCCA testing complements, rather than duplicates, the testing performed by the developer. The testing

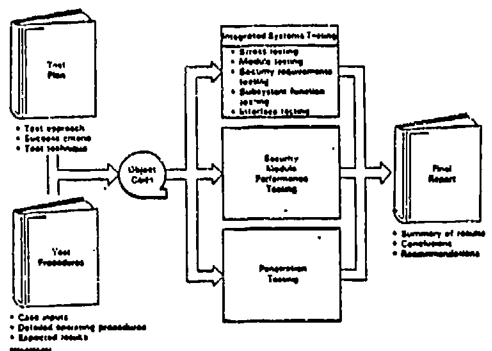


Figure 6. CSCCA Testing

is developed in a methodical manner from the results of the requirements analysis which determined if (and to what degree) the systems requirements reflect the security requirements. High level test specifications are written for each security requirement. These tests are designed to demonstrate the correctness and effectiveness of each security requirement.

CSCCA also uses stress testing to examine critical security functions during program execution when many demands are placed on the system. These stress and/or penetration tests will be developed from the results of our previous analysis using a fault tree hypotheses. The fault tree hypothesis, Figure 7, postulate techniques that could be used to exploit the system weaknesses and circumvent the system security measures. The techniques are translated into system-specific tests to provide or disprove the hypothesis. Since it is impractical and frequently impossible to test all combinations of inputs and program paths, the testing performed must be sufficiently representative of the entire spectrum of possible conditions to establish confidence in the security controls of the system. In general, software development methodologies do not establish criteria for meeting this goal. However, the design analysis and code analysis activities in CSCCA serve to generate these criteria.

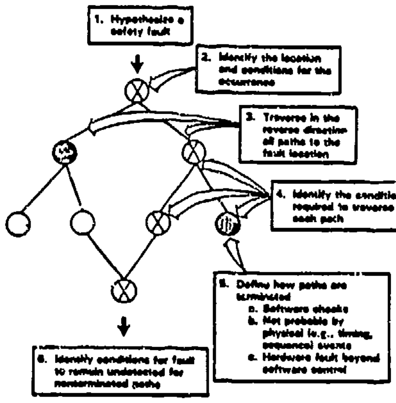

1. Hypothesize a safety fault
2. Identify the location and conditions for the occurrence
3. Traverse in the reverse direction all paths to the fault location
4. Identify the conditions required to traverse each path
5. Define how paths are terminated
a. Software checks
b. Not probable by physical (e.g., timing, sequencal events
c. Hardware fault beyond software control
6. Identify conditions for fault to remain undetected for nonterminated paths

Figure 7. Fault Tree Hypothesis

The CSCCA effort is aided considerably by the use of carefully selected software tools which provide reliable, cost-effective adjuncts to manual analysis techniques. Tools significantly increase the productivity and value of a security evaluation effort. Both static and dynamic tools can be used. Static analysis tools do not require program execution; metric analyzers, requirements tracers, dataflow analyzers, and cross reference generators are typical of the tools in this category. Some operate on requirements and design information supplied by the analyst, while others are used to help analyze program source or object code. Dynamic analysis tools include test drivers, execution monitors, real-time analyzers, data reduction tools, and simulators. These tools are used to control program execution, extract meaningful information during program operation, analyze program results, and model the environment external to the operating program.

CSCCA is concluded with a system validation and recommendation for certification. The purpose of validation is to provide final assurance that the as-bulit system satisfies the specified system security requirements and meets critical security objectives. Validation provides an end-to-end evaluation of the software deliverables against the security requirements, the security model, and the critical security objectives. Since it is common for numerous program patches and other minor software changes to be made during the final stages of system test and integration, it is crucial to the success of the CSCCA process that a final validation of the entire system follow delivery of the final code.

The final CSCCA activity is a certification of the object program delivered to the program office. This two-step process includes: (1) an internal verification that the final-version source and object tapes correspond to one another and match the work files used in performing CSCCA, and (2) a certification demonstration conducted to confirm the object tape delivered to the program office is identical to the object tape on which validation was performed.

4. Summary

The last decade has brought an increased understanding of security-specific technical issues inherent in the development of computer systems. These technical issues include:

- o Utilization of a security model that accurately reflects the security requirements.
- o Use of a correctly implemented tamperproof security kernel.
- o Rigid adherence to modern software engineering standards throughout all phases of system development.

There now remains one significant technical issue, that is the need for a rigorous, independent, and objective assurance procedure to meet critical security objectives.

When the need for reliably secure computer systems first arose in the early 1970s, it was believed possible to design and implement computer-assisted assurance procedures which would be able to verify the correctness of a program with mathematical certainty. The mathematical theory behind such formal verification procedures is mostly understood; but due to the complexities of large systems, the use of formal verification proofs to verify correctness of actual programs has been very limited. It is unclear when this technology will be applicable to a large computer system.

CSCCA is a well proven rigorous technique. CSCCA carefully tailors IV&V technology to the special issues inherent in developing secure computer systems. It is a blending of rigorous computer-assisted review, analysis, testing, and evaluation activities designed to provide objective assurance that a system meets its critical security objectives. Computer Security Cross Check Analysis—performed in conjunction with a software development process employing modern software engineering principles—is the best, most cost-effective, practical way to realize state-of-the-art computer security in mission critical computer systems.

The authors wish to acknowledge Jerry W. Mersky and Bruce H. Wetts for their contribution in preparing this paper.

A1 ASSURANCE FOR AN INTERNET SYSTEM: DOING THE JOB

P. C. Baker, G. W. Dinolt, J. W. Freeman, M. Krenzin, and R. B. Neely

Ford Aerospace and Communications Corporation

Colorado Springs, Colorado

This research was sponsored in part by the USAF Rome Air Development Center under the Multinet Gateway Program, contract number F30602-86-C-0138.

A project to certify a multilevel secure internet device has served as a vehicle to address several design issues for secure communications systems. These issues have been the subject of intense discussion by the security community in recent years. This paper specifically covers three of the issues. Our approach to providing assurance for communications systems not only is doing the job for Multinet Gateway, but also applies to a broad class of communications and other systems.

INTRODUCTION

We are currently developing an internet device that, in conjunction with other internet components, provides a datagram service. In providing that service, it also provides security protection mechanisms to prevent the compromise of sensitive information. The security protection mechanisms are to be evaluated in an internet environment using as a basis the Trusted Computer System Evaluation Criteria (TCSEC) at the A1 level [5]. TEMPEST and COMSEC certification are also being accomplished. To accomplish this, extensive use of rigorous and formal methods are being employed. The paper also describes those methods in terms of objectives, motivation, utility and application to the internet device and its environment. This paper describes our approach to the development of such a secure distributed system device and some of the issues such an effort raises.

The following list summarizes the issues covered in the process of the Multinet Gateway certification project:

- life-cycle issues: re-verification cost and risk
- documentation relationships: linking formal, traditional, and other documentation
- requirements analysis: describing requirements for enhanced traceability and identifying important relationships among requirements
- obtaining assurances from different sources: logically combining constraints on environment and component behavior to satisfy qualitative assurance requirements
- use of rigor: avoiding the all-or-nothing effect often associated with formal methods
- flexibility of system types: focus on communications systems
- secure system architecture: characterization of a distributed TCB via the more general concept of a "constraint monitor"
- specification target: what is being specified and what about the specification is to be verified
- decomposition and abstraction: examination of the fundamental ways of relating components in a complex system

This paper focuses on the latter three issues. Elaborated as questions, those issues are:

- **Secure system architecture:** How should one characterize the analog of the trusted computing base (TCB) for a communications system (or any distributed system)? What are the consequences of directly extending the TCB concept to general systems? Have some fundamental issues been overlooked by such a direct extension? [16]
- **Specification target:** What is the target of the specification, verification and (eventually) certification process? What is it that a formal specification must describe, and is it only to be a formal <u>top level</u> specification of the target? [4]
- **Decomposition and abstraction:** How are these concepts related? Are they merely inverses of each other? If not, how do we know when to use either?

The remaining listed issues are discussed in "Rigorous Integration of Sources of Assurance" [14] and "Results in the Development of a Multilevel Secure Network" [6].

The remainder of this paper is organized as follows: The second section describes the Multinet Gateway as an internet device and its environment in terms of design constraints, design objectives and security concepts. The third section describes the various development mechanisms that are being used in the Multinet Gateway development. Included in this section are the motivation for newly developed tools (trust domains) and concepts (constraint monitors). The Gypsy specification language is identified in this section to show how it integrates with the development mechanisms. The fourth section describes how we are applying these development mechanisms in order to specify the necessary security characteristics of the Multinet Gateway internet device. The paper concludes by relating our results to previous computer security literature.

SYSTEM DESIGN CONSIDERATIONS

The Multinet Gateway System (MGS) is composed of gateways and networks. It provides an internet protocol based datagram service that permits delivery of datagrams between source and destination hosts, subject to the DoD security policy on the transfer of classified information. The MGS will be able to connect to networks that carry data at one or more security levels. The current Multinet Gateway certification program has the goal of providing an A1 level of security assurance for the gateway as an internet device. The certification work includes not only formal specification and verification of the security-critical software, but also COMSEC and TEMPEST certification. The protection mechanisms for personnel, TEMPEST, and COMSEC as well as the procedural mechanisms are all applicable to the MGS, but are not emphasized in this paper.

Design Constraints

Inherent in the design of any system are the constraints that limit the design choices:

- Access to data: The MGS provides switching services that employ the lower-layer protocols in terms of the OSI Reference Model [11]. These lower-layer protocols do not require access to user data in order to perform their particular functions. The primary data objects to be protected are user data that are encapsulated in various protocol layers.

- Well-Defined data structures: The only way that users can access the MGS is by formatting their data in accordance with the specifications of these lower-layer protocols. These formats are well-defined in terms protocol header fields, data fields and trailer fields. In particular, these sets of protocols provide a means of unambiguously determining the location of the security label in the protocol header (if appropriate) and determining the demarcation between the protocol header/trailer fields and the user data.

- External Systems: The MGS is intended to function in the context of a larger system that includes hosts and networks that are external to it. The security properties of the MGS must be considered in conjunction with these external components. The security of the combined components including the MGS may be affected by the security attributes and security properties of these external components.

- Distributed system: The MGS is intended to provide interoperability between geographically dispersed networks, and is therefore itself a geographically dispersed system with specific functionality to be distributed.

- Formal specification and verification: Formal specification and verification techniques are to be used to provide an increased level of assurance that the mechanisms whose functions are to prevent unauthorized disclosure of sensitive information are correctly implemented.

Design Objectives

The preceding design constraints are considered requirements that must be met by any design approach for the MGS. In addition to those constraints, there are additional design objectives which have further limited the potential approaches.

- Security and protocol processes: Minimize the security-critical portions of the protocol processing. Protocol design, development, and implementation is currently undergoing rapid change as the technology evolves. Limiting the amount of security trustworthiness required of these protocols will significantly reduce the MGS security recertification effort required as protocols are added, deleted, or modified.

- Common Protection Mechanism: Provide security protection mechanisms that can be uniformly applied to each of the distributed components that make up the MGS.

- Security Labels: Provide a means of associating or tagging all (user level) data processed by the MGS with a security label.

- Certification of the COMSEC Integration: Ensure that the design of the embedded encryption function and the integration of the device are accomplished in a consistent and verifiable manner. Related issues include those of defining an appropriate policy, model and specifications, which capture the security-critical functionality and desired security properties of the COMSEC equipment.

- Levels of abstraction: Use levels of abstraction as part of the design process for three purposes. It will simplify the task of evaluating the security properties of the system using a top down approach. It can modularize the specifications so that changes can be easily encorporated without complete re-verification. It can simplify the verification process.

Multinet Gateway System Security Concepts

The basic Multinet Gateway System security concepts are:

- The definition of the system boundary of the MGS. The definition of the system boundary is to incorporate the notion of various security perimeters.

- Definition of all information units and access control of such information units crossing the boundary.

- The control of how data units are manipulated while inside the boundary.

These concepts must be placed in context of the physical realities of the Multinet Gateway System. The Multinet Gateway System consists of a set of geographically dispersed Multinet Gateway Nodes. The nodes are connected to client hosts by networks termed "End Networks", and are connected to each other by networks, termed "Transport Networks". A network that provides both end and transport services is termed a "Dual Network". Figure 1 illustrates these relationships. The MGS boundary establishes the point where security responsibilities start or end. The Multinet Gateway System boundary, in terms of Figure 1 is where the End Network connects to the MG Node. The Transport Networks are considered inside the MGS boundary.

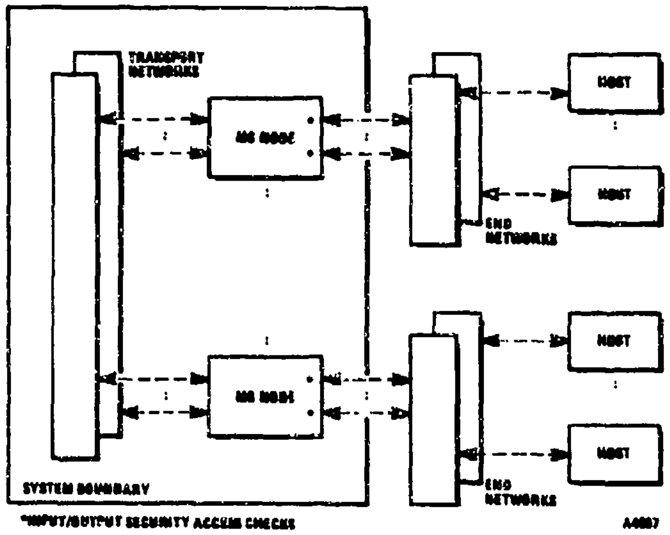


Figure 1. Multinet Gateway System

Given the MGS boundary, mechanisms are required to control the access into and out of the MGS. The location of the access control is shown in the figure.

The concepts underlying the computer security protection mechanisms are the following:

- Within the MGS a security label is associated with every information unit (datagram or other protocol unit).
- An input security access check is performed on every information unit entering the MGS.
- An output security access check is performed on every information unit leaving the MGS.
- While in the MGS, information units will not be subject to unauthorized disclosure.
- While in the MGS, the security label associated with a information unit will not be corrupted.

A major source of assurance that these security considerations are met is the Secure Communications Support System (SCSS). The SCSS consists of the software, executing on each gateway processor, that supports the security controls.

DEVELOPMENT MECHANISMS

The development mechanisms we are using are summarized as follows:

Trust domains: A means of partitioning a distributed system and expressing particular constraint relationships.

Constraint monitors: A means (in conjunction with trust domains) of determining what minimum constraints are necessary to maintain the security of the system.

Gypsy Verification Environment (GVE): A means of expressing the design and implementation and proving assertions that are expressions of the constraints.

Documentation: An integrated way of describing the system and its security attributes.

Trust Domains

The distributed nature of the Multinet Gateway environment led us to consider a rigorous means of describing such systems. The result was the development of the concept of trust domains [15]. Briefly, trust domain analysis provides a high level description language which can be used to describe the partitioning of systems into nodes and links. The language provides a way of describing how nodes and links are interconnected, how each can be broken up into subordinate nodes and links, and how each can be constrained at the associated interface. These language capabilities can then be used to focus on the security properties of the system without being tied to the semantics of a given specification language, such as Gypsy.

Constraint Monitors

The trusted computing base (TCB) as defined in the Trusted Computer System Evaluation Criteria (TCSEC) is that software and hardware used to enforce the system security policy. One would like to minimize the amount of such software/hardware because it is expensive to produce for A1 systems.

Based on trust domains, we have developed the notion of a "constraint monitor" to aid in the determination of exactly what portions of a system are trusted and what they are trusted to do. The term constraint monitor is used to replace such terms as "kernel", "reference monitor", "trusted process", etc, to emphasize that in a distributed system not all functions or components will have the same type of behavioral requirements (trust).

Gypsy Verification Environment

The GVE provides a capability to specify and verify (via proof techniques) correctness attributes of systems and their constituent components, including programs. The GVE consists of a programming language, Gypsy, a verification condition generator, a theorem prover, and tools for maintaining the verification state of the current version of the system under development. The programming language includes mechanisms for specifying properties to be proved about systems, subsystems and programs and their interrelationships.

A significant advantage of Gypsy is its ability to specify and verify systems which have concurrent processes, which is particularly useful for describing distributed systems. The GVE is described in detail in "Using the Gypsy Methodology" [8].

Documentation

Given the various design tools and concepts mentioned above, there is the need to provide some way of conveying the design information to an evaluator in a consistent and understandable form. Because of the complexity of typical distributed communication systems, their design must be described in several abstraction levels, in terms of specific properties that are to be emphasized, in mappings of how the specific properties are related to the overall functionality of the system, and finally how the various components are implemented in iron and executing code. Unfortunately, such a universal description language does not exist. As a (hopefully interim) substitute, we have developed procedures to provide correspondence between the available specialized description languages. A document termed the System Security Description (SSD) is used to present the various descriptions and their associated correspondences.

The system and security architecture is presented in the SSD via a four-fold representation: an implementation, a functional, a trust domain and a Gypsy summary description.

The first description is an implementation description. It includes the physical location of the components, how they are interconnected, the allocation of functions to software, firmware, and hardware and the protection mechanisms.

The second is a description of the functionality of the MGS. It describes the types of functional processing and the associated relationships. This functional description provides a basis for another description that is a more rigorous formulation of specific security requirements and their relationships.

This third description allows a demonstration that the MGS security policy is satisfied without being tied to the semantics of a given formal specification language. This description is given via the concept of "trust domains" and the trust domain description language.

The fourth description is a summary description of the Gypsy specification of the MGS, with its focus on the required security attributes. Although the formal specification itself presents a description of the MGS and its security attributes without any implementation or procedural constraints, the SSD presents a class of implementation constraints that are consistent with the formal specification of the MGS and to which the MGS implementation belongs.

The final part of the SSD provides the mappings between the four descriptions to show how they correspond to one another. The mappings are represented and validated informally.

Together, these descriptions are intended to comprise what is meant by a "descriptive specification" as called out by the TCSEC. The descriptions permit a clearer presentation of the many aspects of the design and give a reader a more complete view of the overall design. These documentation relationships and the certification approach for Multinet Gateway development are discussed in further detail in "Results in the Development of a Multilevel Secure Network" [6].

ISSUE RESOLUTION

In this section we show how we are actually building the MGS using the development mechanisms presented previously.

Abstraction Layering and Decomposition

For the purposes of this development, it seems appropriate to view abstraction and decomposition in terms of dependency relations. The concept is that elements of a set "upper" components depend on the elements of a set of "lower" components. Such a dependency, in the Multinet Gateway security design, is both in terms of function and constraints. The distinction between an abstraction relationship and a decomposition relationship is that with a decomposition, the relation is a many-to-one mapping from the lower component set to the upper component set. With an abstraction, the relation is typically many-to-many.

The many-to-one versus many-to-many distinction is significant in the actual development process. A many-to-one (decomposition) mapping requires much less "bookkeeping," and is generally supported by more automatic tools (such as the syntax of the Gypsy language) than the many-to-many (abstraction) mapping. Thus, it is generally prudent to apply decomposition wherever some overriding factor does not prohibit it.

We chose to incorporate the abstraction mapping in the development process based on the following factors:

- the need to structure the formal specification in a way that relates easily with the policy model;
- the need to assure certification plug compatibility of interfaces, i.e., reusability of the verification and certification for certain trust domains; and
- the need for a direct relation between the formal specification and various "architectural reference points" in the design.

By an architecture reference point, we mean a significant aspect of the system architecture that is taken as given and so must be reflected by not only the system implementation, but also by the formal description of the system. For example, in a system of network gateways, and architectural reference point might include specific aspects of the geographical separation of the gateway nodes. The idea of an architectural reference point is that it constrains the allowed design space of the system.

In order to reflect the policy model in the formal specification of the system, it was useful to keep the virtual secure transport, acceptance and delivery functions highly visible. Also, it was decided that the SCSS external interfaces should be explicitly represented in the formal specification to assure its certification plug compatibility. The primary formal constraints (security policy model) would not have necessitated that representation, nor was the system architecture a significant factor. But the SCSS is expected to be reusable, and its reuse needs to have a minimal verification impact.

The combination of the need to reflect the policy model as it is in the formal specification and the desire to make visible the SCSS with its external interfaces disallowed direct decomposition from the system interface to the SCSS interfaces. Thus, an abstraction mapping was necessary at the SCSS interface, with the two layers related by the constraints satisfied by the SCSS.

In summary, the decision to include an abstraction mapping was the result of significant iterations of architectural structuring during the past two years.

An Examination of the TCB Concept

When applying the concept of a TCB within the context of a distributed system, there are some resulting issues that can be quite confusing. A key to avoiding such confusion is to look at distributed or otherwise complex systems in ways tailored to the systems and their environments [13]. Before we can know how to apply such a concept, we must first ask, "What must we prove about what?". First, in a deeply structured system, it becomes more clear that the system as a whole is the object about which adherence to the security policy must be proved, and not just some "security-critical" software (which might be called the TCB). Second, as we examine internal, derived security requirements, we see more subsystems that are not strictly a "TCB" or parts of a "TCB" about which those derived requirements must be proved. Third, according to the TCSEC's definition, the TCB encompasses more than developed software. For example, part of the hardware of a system may be part of the TCB. Thus, it may not be appropriate to prove some of the TCB characteristics, for various reasons. Fourth, different, separated parts of the TCB may be different in function and/or interface, and must somehow be related. Further, some sets of separated parts of the TCB may be essentially identical except for their contexts, and so should not require separate proofs of their constraints. Fifth, a distributed system requires explicit attention to the channels (links) as well as to the computational components (nodes). Links may be as complex as nodes; in fact, either may contain the other. Trustworthiness of links may have to be demonstrated or assumed, as with nodes. As a special case, trustworthy links must be available to allow communication among separated components of the TCB.

The consequence of these points is that distributed systems inherently have structure. Ignoring such structure by assuming a "system TCB" without substantiation detracts from any confidence associated in the assurance mechanisms. It is more beneficial if there are ways to deal with such underlying inherent structure.

A more useful view is an approach analogous to the operating system monitor concept [10]. The paradigm for that approach is depicted in Figure 2. The concept of a TCB has been replaced with "constraint monitor," since the paradigm is to be applied at any place in the system structure where a constraint is to be demonstrated. In particular, it is not to be applied just at the system security policy level. The paradigm thereby accounts for both the traditional TCB concept, as well as more general concepts.

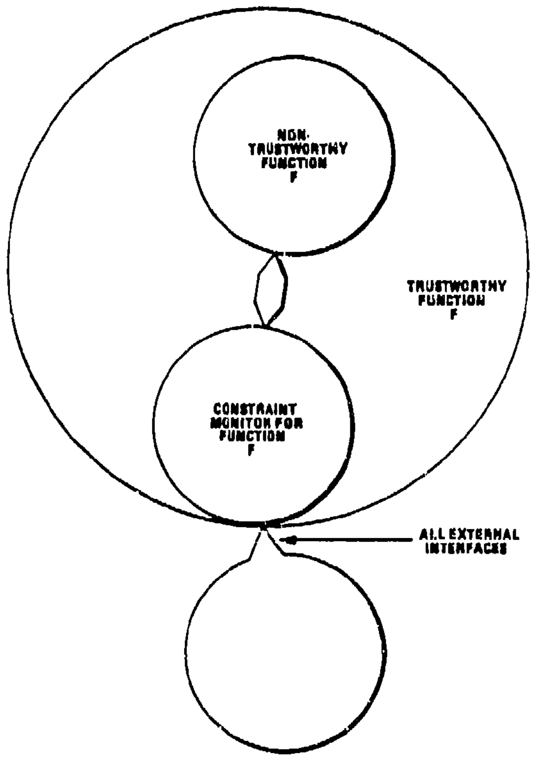


Figure 2. Constraint Monitor Paradigm

As an example, let the "non-trustworthy" function F, identified in the figure, be the amalgamation of all the application (user) functions to be performed on a secure operating system. The constraint monitor is then just the software portion of the TCB, as traditionally viewed. The "trustworthy" function F' appears to users (connected via the "external interfaces") as the system itself.

We now apply the constraint monitor paradigm to a distributed system, such as the MGS. In a first application of the paradigm, the SCSS is the constraint monitor itself. The non-trustworthy functions are the protocol functions, which we are demonstrating need not be trusted with respect to the security policy. That demonstration involves showing that the SCSS constrains the functions appropriately, and that the only interfaces to the functions involve the SCSS. The set of "trustworthy functions" is then the software present on a particular processor. All external interfaces to such a set of software pass directly to the SCSS.

The paradigm is used successively in the MGS structure until one arrives at a trustworthy internet device, the Multinet Gateway node. That device, in its context, is another example of a constraint monitor. As with the SCSS, it is multiply instantiated within the MGS. The non-trustworthy function in this context is the set of transport networks by which the internet devices communicate with one another. The paradigm is applicable here, since all external interfaces, i.e., to client networks, are via the internet devices. As a result, the MGS is a trustworthy version of the transport function.

Formal Specification Structure

The Gypsy language representation of the MGS includes the MGS interface, the MGS environment, and certain subsystems, particularly the Multinet Gateway node and the SCSS. The SCSS and subordinate components are represented in an abstraction layer separate from the rest of the system. The reason for this is that the MGS Outer (System) abstraction follows functional architectural reference points, while the SCSS abstraction follows implementation architectural reference points. Each is described in this section along with an explanation of their interrelationships. The previously described concepts are thereby discussed in the context of a given specification language.

MGS Outer (System) Description

The MGS Outer (System) abstraction relates functional architectural reference points such as the internet structure, protocol layering and system-level protection structures, e.g., gateway-to-gateway encryption. Figure 3 is a summary of the specification decomposition for the system layer of the MGS. The figure depicts the structure of the Gypsy procedures and cobegins.

```
MGS Environment
  Cobegin:
    User Delivery
    User Acceptance
    MGS
      Cobegin:
        MG Acceptance
        MG Delivery
        MG Derive
          Cobegin:
            E*3 Delivery
            E*3 Acceptance
            Node Derive
            E*3 Derive
              Cobegin:
                E-Box
                D-Box
                NAP Derive
                 Cobegin:
                   Label Create
                   Label Delete
                   Transport Derive
                     Cobegin:
                       Transport Deliv
                       External Transport
                       Transport Acceptance
```

Figure 3. Gypsy Specification Tree for System Layer

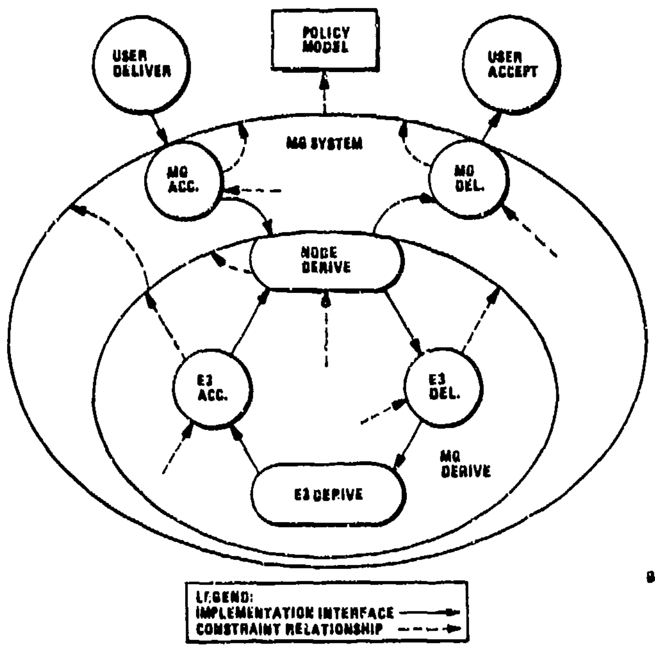


Figure 4. Specification Diagram for System Layer

Figure 4 shows the upper portion of the system Gypsy specification tree pictorially, with the connections (Gypsy buffers) as arrows. This figure further contains broken arrows that indicate certain constraints assumed by some components and satisfied by others. The constraints are represented in Gypsy by entry, exit, and block conditions, along with supporting specification functions as constraints. Note that while some of the constraints have both "satisfiers" (tail end) and "assumers" (head end), other constraints have an unconnected tail end, and so are attached only to "assumers." Such constraints, in the system layer, are treated as boundary conditions. It is only via the formal mapping between this layer and the SCSS layer that those constraints are demonstrated to be met (cf. "Mapping" paragraph, below).

MGS Inner (SCSS) Description

The MGS Inner (SCSS) abstraction relates architectural reference points that are implementation specific. For example, hardware, software, abstractions, the design, performance and modularization strategies. Figure 5 is a summary of the of the specification decomposition for the SCSS layer of the MGS. The figure depicts the structure of the Gypsy procedures and cobegins.

```
SCSS Environment
  Cobegin:
    Actuator
    SCSS
     Cobegin
      Call Decoder
      Call Encoder
      Local Functions
      Prodige
      MMI Service
      Net Service
      ITP Service
      COMSEC Service
      E3 Service
      IPC Switch
```

Figure 5. Gypsy Specification Tree for SCSS Layer

Figure 6 shows the SCSS Gypsy specification tree pictorially, using the same representations as in Figure 4. In this figure, broken arrows (constraints) either have both ends connected, or else have an unconnected head end, and so are attached only to "satisfiers." Such constraints need not be assumed by any domains in the SCSS layer, but rather are intended to satisfy boundary condition constraints in the system layer. That connection of constraints is accomplished via the formal mapping described in the "Mapping" paragraph, below. The abstraction layer that underlies the SCSS layer is the Gypsy computational model. The SCSS assumes that the Gypsy layer operates correctly according to Gypsy semantics. The Gypsy layer in turn assumes that the hardware operates according to the security-relevant constraints that will be specified for it.

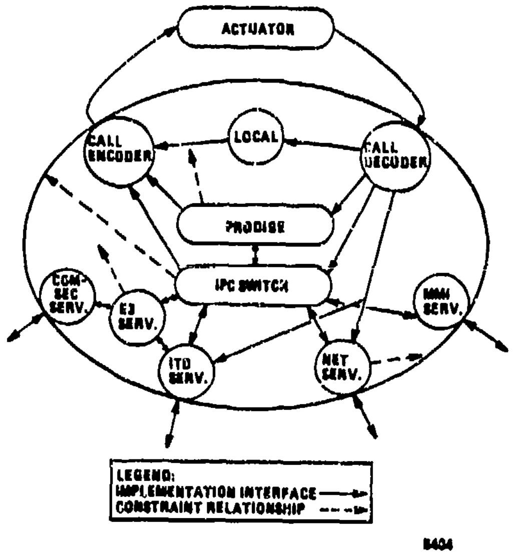


Figure 6. Specification Diagram for SCSS Layer

At completion, the SCSS formal decomposition and the SCSS implementation software will be in a one-to-one relationship in terms of modules. It will be given to a level of detail sufficient that within each module, the determination of whether the implementation corresponds to the specification constraints will be simpler and less prone to error than previous development efforts [15].

Mapping Between the System and SCSS Abstractions

Four fundamental concepts characterize the mapping between the SCSS and system layers:

- constraints, rather than structures, are mapped;
- constraints of the system layer that must be satisfied by the SCSS are left as boundary conditions, assertions that may be assumed but whose proof is deferred;
- a set of constraints intended to map to the unproved system constraints are proved within the SCSS layer specification; and
- one or more Gypsy scopes are provided containing functions to map the proved SCSS constraints to the unproved system constraints.

The primary mappings are depicted in Figure 7. The figure contains a simplified version of Figures 4 and 6. In addition, the figure contains "junction boxes" in the center, which represent the mapping functions that allow the upper and lower versions of the constraints to connect.

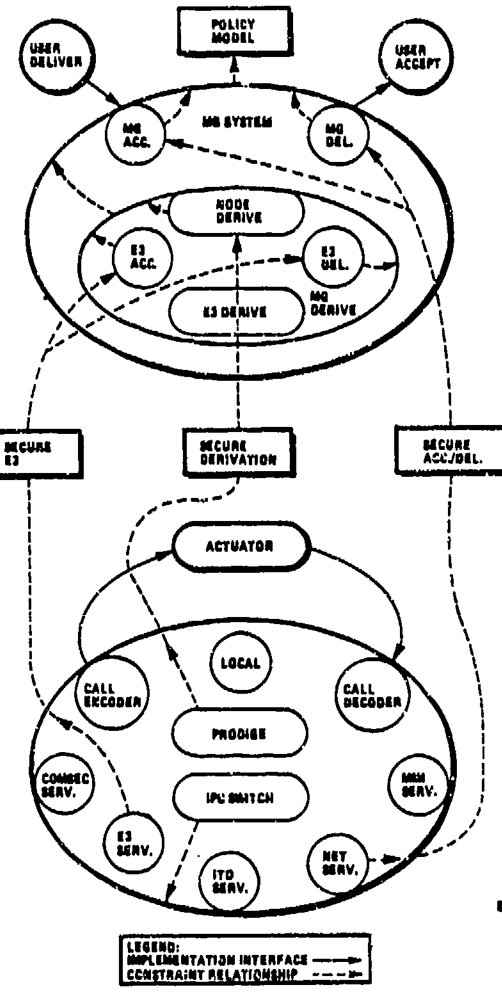


Figure 7. Constraint Mapping Between Abstraction Layers

The mapping functions themselves are of two types: type mappings and constraint function mappings. The type mappings resolve the differences in Gypsy types between the two abstraction layers. The constraint function mappings express the system layer Gypsy functions naming the constraints in terms of the analogous SCSS layer Gypsy functions.

RESULTS AND CONCLUSIONS

Much of the assurance at the A1 level that is based on an application of formal specification and verification techniques is predicated on a formal model of the security policy and a formal specification. It appears that most of the discussion surrounding policy models and formal specifications seems to center on three aspects: the level(s) of abstraction of the policy and system description, the actual target of the description (e.g., entire system, particular components, explicit functionality) and the appropriateness of existing models including specific interpretations of them (e.g., Bell-LaPadula [1], the Military Message System [12], Two-Level Model [7]). This paper asserts that part of the criteria for a model's appropriateness should include whether one's understanding of the targeted system is actually increased (both of the design and what is verified) rather than forcing an interpretation of a model such as the Bell-LaPadula model.

For the MGS, the security policy model is an abstract model with a structure for explicit instantiations to provide a collection of distinct policies (which may become constraints on specific portions of the MGS) that are consistent with the more abstract policy model. In terms of the previously identified aspects, the target of the policy model and the formal specification is the entire MGS as an internet. The formal specification provides a description of the internet system down to particular routines executing on particular processors within a given node of the MGS. Although the rules of the Bell-LaPadula model are applicable to the environment of the MGS, they are less so to the datagram service provided by the MGS and for the lower protocol layers. This is in concurrence with an observation made by Denning at the 1985 Invitational Workshop on Network Security [3]. The observation is that those viewing a network as a distributed resource manager (higher levels of the ISO model) find less trouble with the TCSEC than those who view a network as a communications subnet (lower levels of the ISO model). The Multinet Gateway modeling approach has provided a means, via the trust domain representation and the formal specification approach, to tie these views together without forcing interpretations that do not aid the overall understanding of the system. A key aspect of the approach being taken is that a effective framework for the various descriptions is provided so that an interpretations of a security requirements (according to the TCSEC) can be analyzed in a rigorous manner.

Bell [2] raised the issue of the exact manner of intertwining the development of the formal top-level specification (FTLS) and descriptive top-level specification (DTLS). The intertwining of the FTLS and DTLS issue is resolved within the Multinet Gateway development process via the production of the four types of descriptions discussed in the "DEVELOPMENT MECHANISMS" section and the relationships among them. A consequence of this is a clearer design certification chain down to the implementation of the security mechanisms within the enhanced ADM and their relationship to the whole MGS.

An additional contribution of our approach is a way of addressing a relatively hard problem in the specification of system-wide and component specific aspects as described by Schaefer and Bell. The "... problem is how to be assured that the collection of component specifications support, are consistent with, and actualize the system-level specification ...."[17]. They correctly point out the difficulty of handling such a process in present formal verification systems. The objective is to provide an association between an expression of the requirements and design at one level and derived requirements and design at another level. As part of the Multinet approach, such an association between levels of system description is achieved.

Finally, based on initial results, it appears that the approach outlined here offers a way of lowering the overall cost of the specifi ation and verification process. This is due to the way that a reasonable and reusable problem domain can be described in a consistent manner in shorter time and communicated to people not all of whom are experts in a given formal specification language. This is consistent with the observations and suggestions offered by Good[9]. The approach also helps reduce the technical risk in an application of the formal specification and verification technology.

In conclusion, this paper has examined some security issues that have been under discussion in the security community in recent times. Based on the work already accomplished for the certification of an Multinet Gateway internet device, an analysis and assessment of the issues has been made and results discussed. Much of the basis for the assessment is due to the engineering work necessary to produce an advanced development model of an internet device and to demonstrate the feasibility of the design concepts. This was accomplished by building such a device, by having a parallel effort to prototype some of the technical aspects for the security certification of the device and by incorporating the prototype results into the formal specification and verification of the system. Questions raised in the literature and calls for research in specific areas have been addressed by the sharing of results obtained in the given context, Multinet Gateway. Although the focus presented here is but one way to consider the underlying issues, it is felt that the results are applicable to a wider range of systems under development or being planned.

REFERENCES

1. D. Bell and L. LaPadula, "Secure Computer System: Unified Exposition and Multics Interpretation," MTR-2997, MITRE Corp., Bedford, Mass., July 1975.

2. D. Bell, "Working Towards A1 Certification," Proc. 7th DOD/NBS Computer Security Conference, Gaithersburg, Md., 24-26 Sept. 1984, pp. 24-29.

3. D. Denning, "A Position Statement on Network Security," Proc. Invitational Workshop on Network Security, New Orleans, La., March 1985, pp. 4.47-4.56.

4. G. Dinolt, "Security Policies and Models for Computer Networks" Proc. Invitational Workshop on Network Security, New Orleans, La., March 1985, pp. 2.39-2.48.

5. DoD Computer Security Center, "Trusted Computer System Evaluation Criteria," CSC-STD-001-83, 1983.

6. J. Freeman and R. Neely, "Results in the Development of a Multilevel Secure Network," Proc. Armed Forces Communications and Electronics Association (Philadelphia Chapter) Physical and Electronic Security Symposium and Exposition, Philadelphia, Pa., August 1986.

7. J. Glasgow and G. MacEwen, "A Two-Level Security Model For a Secure Network," Proc. 8th National Computer Security Conference, Gaithersburg, Md., 30 Sept.- 3 Oct. 1985, pp. 56-63.

8. D. Good, B. DeVito and M. Smith, "Using the Gypsy Methodology," draft Technical Report, Institute for Computing Science, Univ. of Texas at Austin, Austin, Texas, June 1984.

9. D. Good, "Reusable Problem Domain Theories," Technical Report 31, Institute for Computing Science, Univ. of Texas at Austin, Austin, Texas, September 1982.

10. C. Hoare, "Monitors: An Operating System Structuring Concept," Communications of the ACM, ACM, vol. 17, no. 10, Oct. 1974, pp. 549-557.

11. International Standards Organization, "CCITT Red Book, Data Communications Networks, Open Systems Interconnection (OSI) System Description Techniques," VIIIth Plenary Assembly, CCITT, Recommendations x.200-x.250, vol. VIII-Fascicle VIII.5, October 1984.

12. C. Landwehr, "A Security Model for Military Message Systems," Naval Research Laboratory, Washington, D.C., May 1984.

13. C. Landwehr and H. Lubbes, "Determining Security Requirements for Complex Systems with the Orange Book," Proc. 8th National Computer Security Conference, Gaithersburg, Md., 30 Sept.- 3 Oct. 1985, pp. 156-162.

14. R. Neely and J. Freeman, "Rigorous Integration of Sources of Assurance," Proc. IEEE (Washington Chapter) Symposium on Computer Assurance: Software Safety and Process Security, Washington, D. C., July 1986.

15. R. Neely and J. Freeman, "Structuring Systems for Verification," Proc. IEEE Symposium on Security and Privacy, Oakland, Calif., 22-24 April 1985, pp. 2-13.

16. D. Nessett, "Factors Affecting Distributed System Security," Proc. IEEE Symposium on Security and Privacy, Oakland, Calif., 7-9 April 1986, pp. 204-222.

17. M. Schaefer and D. Bell, "Network Security Assurance," Proc. National Computer Security Conference, Gaithersburg, Md., 30 Sept.- 3 Oct. 1985, pp. 64-69.

ON THE INTERACTIONS OF SECURITY AND FAULT-TOLERANCE

R. Turn
J. Habibi

Department of Computer Science
California State University, Northridge
Northridge, CA 91330

ABSTRACT

Security and fault-tolerance are critical requirements in many distributed systems, especially in those supporting real time applications. For fault-tolerance, redundancy techniques are being applied in hardware or software to attain fault masking and graceful degradation capabilities. For security, various approaches are being proposed for distributed system use. This paper examines interactions between security and fault-tolerance generically and with the help of a case study. It concludes that, for fault-tolerant security, hardware implemented redundancy techniques are to be preferred over software implementations.

INTRODUCTION

A recent trend in computer system architectures is to design and implement distributed systems where sets of processing units (PUs) (each with its own private memory and I/O devices), global memories, and data bases are interconnected by local area networks (LANs). The distributed system architecture has several useful features: the processing power of the system can be increased (or decreased) by adding PUs (or removing them); availability and reliability are enhanced by virtue of multiple PUs, data bases, etc. such that these units can be used to back up each other; and cost is lower, as compared to using a mainframe computer with the same processing power (if such a computer is available at all).

Distributed processing is the preferred architecture in a variety of real-time applications, such as command-control systems for the military, process control systems, air traffic control, and others. These systems must be highly reliable and available, and they are also likely to contain or process sensitive information, and require that the integrity of operational data and programs be protected. In these systems, data security and integrity become additional design requirements. Some of these systems serve large communities of users who have different security clearances or need to know. For efficient resource-sharing operation, multilevel security (MLS) would be required in these systems.

Given the requirements for fault-tolerance, graceful degradation, and data security, a number of questions about the interactions of these requirements arise and need to be resolved:

Are the techniques for achieving fault-tolerance and data security fully compatible? If not, what are the problems, and how can they be resolved? What tradeoffs are available?

How does the architectural design for fault-tolerance impact the design for security, and vice versa? How can the designs be made compatible?

Can data security be gracefully degrading? Can gracefully degrading systems be data secure? Is there a difference in achieving each?

This paper addresses the above questions generically in terms of the suitability of various strategies and techniques of fault-tolerance in sytems where security is also required. A brief illustration of implementation problems is provided by examining MuTEAM, an existing, experimental distributed system with fault-tolerance, from the point of view of rendering it multilevel secure.

FAULT-TOLERANCE TECHNIQUES

In this paper, "security" is defined in terms of the implementation and enforcement of the DoD security policy according to the DoD trusted computer system evaluation criteria[1,2]. Since the design requirements and techniques for security are well-known they need no further explanation in this paper.

"Fault tolerance" is defined as the capability of a system to function correctly according to its design specifications despite of the presence of transient or permanent faults[3,4]. Such faults may occur in the hardware for various reasons, and they may be present in software in the form of undetected "bugs" created in the design or programming. Their effects are "errors" in data values and/or in program behavior. Indeed, one may view security techniques as a subset of fault-tolerance since they are also attempting to handle faults, but these faults may be deliberately introduced and they tend to be very complex. Fault-tolerance techniques tend to be less known and, thus, a brief summary of these is provided below.

Fault tolerance is based on the use of "spacial" redundancy (replicated modules) or "temporal" redundancy (repeated computations). Spacial redundancy is also known as "protective redundancy" -- the occurrence of certain faults is masked entirely by use of redundant hardware or software. A specified number (but a minority) of the redundant units may fail without affecting the correctness the results or the system operation. That is, faults remain invisible.

An example of protective redundancy in hardware, is N-modular redundancy (where N is odd). N replicated modules perform the same operations in parallel. Their outputs are connected to a majority voting unit of N voters which considers as correct the output from the majority of the modules. Some of the voters in the unit may become faulty themselves, but the set performs correctly if less than half of the voters are faulty.

Another protective redundancy technique is the use of error-correcting codes. These mask the occurrence of a specified number of simultaneous faults (i.e., erroneously inverted bits), or specified patterns of faults, in the encoded data unit. Of course, hardware or software design flaws or erroneous input data values cannot be corrected. These require different approaches, such as the "design diversity" technique[5], and various data integrity techniques, such as reasonableness tests.

Temporal redundancy is also known as "corrective redundancy". Here the occurrence of faults must be detected and diagnosed, before corrective action can be taken (i.e., isolation of the failed unit and activation of a replacement unit), and recovery from failure effects can be initiated. Corrective redundancy is implemented is mainly in software, but hardware units can also be used to enhance performance. More specifically, the following actions and preparations are required.

<u>Fault detection</u>, can be implemented in hardware or software. Hardware techniques include the use of error detection codes, self-cheking circuitry, and comparison of the outputs of replicated modules. Software implemented detection includes computation of checksums and other authentication functions, periodic checking of the hardware functioning, various reasonableness and limit tests applied to data values, repeated evaluation of the same function with the same data, and interspersing testing data with operational data. In distributed systems, testing of processing units (PUs) by other PUs in the system is a sophisticated detection strategy which will be examined later.

<u>Fault diagnosis</u>, is an activity which strives to locate the fault and confine the damage. The use of diagnostic programs is one approach. These may be applied by the local operating system or even from remote diagnostic centers without the knowledge of the users and, thus, raise security issues.

<u>Recovery</u>, is the activity of placing the system back into an error-free state from which normal operation can resume. This includes correcting any erroneous changes made when the system was affected by a fault which had not yet been detected. Typically, such error-free system states are stored periodically for this purpose as "roll-back" states. Correcting erroneous data values is a much more difficult task, unless steps are taken in the system design to provide audit trails for data base updates, or "recovery caches" where changed data values are collected. With these means it is possible to restart the computation and restore the data values at the roll-back point.

<u>Reconfiguration</u> is the activity of removing from the system a failed unit, and replacing it with a a correctly operating one. The replacement unit may have been assigned in advance or may be chosen from a pool of available units. This activity may also include transporting programs and data to new units and removing programs and data from a failed unit.

If recovery and reconfiguration have succeeded in restoring the system into an operational state, but with less than nominal capabilities, the system has "degraded gracefully". The degradation may be in the form of a reduction in the system's performance, memory capacity, number of processors available, and the like. One prerequisite for graceful degradation is redundancy in each type of modules, such that the failure of a module can be handled by the remaining modules of the same type which assume the functions of the failed module.

<u>FAULT-TOLERANT SECURITY</u>

Since the purpose of implementing fault-tolerance is to eliminate the disruptive impacts of faults on the functions being performed, it is of interest to apply fault-tolerance techniques also to the security function. The principal purpose of this function is to make decisions on attempts by a system's subjects' (e.g., users, processes) to gain access to the system's objects (e.g. memory segments, files, programs). The security function is fault-tolerant if, despite of faults in the system, the security decisions correctly enforce the system's security policy, the associated decision support data (i.e., identifications, security labels, access rules) remain correct, no sensitive data are erroneously released, no covert channels are introduced, and no denial of service event takes place. We will now examine the suitability for security purposes of the protective and corrective redundancy approaches to fault-tolerance.

<u>Protective redundancy</u>, implemented in hardware, and the use of error-correcting codes appear to be suitable techniques for fault-tolerant security, provided that certain precautions are taken and the assumptions made about the number and patterns of errors are kept in mind. An important consideration is the granularity of applying protective redundancy, i.e., the size of the replicated modules. The finest granularity is at the individual logic function level. A very coarse granularity is at the individual processor level. In general, the coarser the granularity the more semantic content there is in the module's output such that sensitive information might be extracted.

For example, a permanent fault in the control unit of one of a replicated set of data encryption units may cause data to be released unencrypted, but the majority of encryption units would produce correctly encrypted output and the voters would stop the unencrypted data stream. Thus, a replicated module set must be encapsulated such that only the outputs from the voters are visible from outside the module set. Even then, failure of a voter may still allow the incorrect output to exit from the replicated set. The use of a shared bus to send module outputs to be voted upon elsewhere is clearly not acceptable for security.

Corrective redundancy, as discussed above, involves detecting and diagnosing fault occurrences, disconnecting the faulty modules, and then taking corrective action (e.g., recomputing a program from its last error-free "recovery point"). Recovery also includes informing the recepients of potentially erroneous results of the problem and, after recovery, sending the correct results. As discussed in [3], in a highly interactive distributed system this may create a "domino effect" of correction and recovery activities when recepients of erroneous information have passed it on to others, and all those involved must undergo correction and recovery.

In general, this mode of operation appears to be incompatible with the security requirements, since the errors in security functions, and their consequences, may not be recoverable. For example, if the reference monitor malfunctions and permits writing of files in violation of the *-property, a security compromise may result. Important here is the time interval from fault occurrence to its detection and diagnosis (the error latency time). If this time is sufficiently short (error detection test are performed very frequently), it may be possible to recover without any security compromises having occurred.

However, it may also be feasible to design the system for near-continuous self-testing of its security functions, especially immediately prior to any access control or information release decisions, so that some form of corrective redundancy could still be used. Furthermore, in special subsystems such as local area networks (LANs), corrective redundancy is used in the form of retransmission upon error detection. This may be acceptable from security point of view if the LAN interface units are trusted to reject communications which would violate the security policy.

GRACEFUL DEGRADATION OF SECURITY

A system can degrade gracefully if it can remain operational with diminished capabilities despite of the presence of faults which cannot be masked by fault-tolerant design techniques. A prerequisite is that the hardware modules be replicated so that no single failure could totally disable the system. The question is: Can the security function degrade gracefully?

If we use the DoD security criteria as a model, as proposed in Ref. 6, degradation of security can be viewed in terms of downward migration in the criteria divisions and classes. For example, a failure in the TCB hardware may cause the loss of some security mechanism which would cause the TCB to fail to meet some criterion of its current evaluation division and class, but still meet the criteria of a lower class of this division or of a lower division. Thus, a system's security level may migrate from A1 to B3, to B2, and so forth as faults occur. Correspondingly, the authorized application mode of the system would migrate from multi-level secure (MLS), to controlled mode, to system-high or dedicated modes.

Thus, it appears that in principle graceful degradation of security is a feasible concept. In practice, however, it would be necessary to develop a system design where security mechanisms are implemented modularly with diagnostics available to test each module's correct operation. Upon detecting a failure, there would be a need for rapid containement of all subjects and objects until a new (lower) security level has been determined, decisions have been made about the subjects' authorizations, and all data no longer permitted in the system have been safely removed. How this might be implemented is not yet clear.

SECURE FAULT-TOLERANCE

Implementing security in systems where computational fault-tolerance is also required raises new issues for security. The recent trend is toward the use of corrective redundancy, rather than hardware implemented protective redundancy. As pointed out earlier in this section, this may not be fully compatible with security requirements. The replicated modules, too, tend to be more complex and so the redundancy granularity is coarser. All this makes the system more complex and creates new information flows. Thus, there are more security-related problems to resolve: proving correctness of the design and implementation, and analyzing the system for new information flows and new covert channels.

Software-implemented corrective redundancy is usually based on the "backward recovery" concept -- intermediate results of computations and system state information are stored periodically as "recovery points" and, if error is detected, computation is restarted from the most recent recovery point. Several copies of recovery information may be kept to allow backup of faulty modules. This increases the exposure potential of sensitive information and complicates meeting the security requirements.

To illustrate the secure fault-tolerance problems, we will briefly discuss MuTEAM, a distributed multimicroprocessor system prototype developed in Italy[7-11]. This system is intended to serve as a development vechicle for design methodologies of real time distributed systems applications. It operates in a decentralized control mode without a central supervisory control entity, and it includes an integrated set of fault-tolerance mechanisms in the system programming language, architecture and run-time support. Its main goals are to achieve modularity, expandability and fault-tolerance and to maintain a concurrent processing env.:onment in each processing unit.

The system is organized into a set of clusters of up to 16 computer elements (or "nodes"). These are connected via a "cluster bus", and a "signalling bus". The interconnection topology satisfies the requirement that each pair of nodes is connected by at least one communication path, despite any single link failure. All interprocessor communication is based on message passing using the shared memories at nodes. Access control list technique is used for protecting segments in the shared memory.

MuTEAM's fault-tolerance is based on separate phases of fault detection and diagnosis, reconfiguration, and recovery. A fault in a node is viewed as rendering the node totally faulty and it is disconnected from the rest of the system. A faulty node is isolated from the non-faulty ones by the latter deleting all access permissions of processes in the faulty node. Processes from a faulty node are reallocated to non-faulty nodes based on prior pairing of nodes with their "twins". All these operations have considerable security implications.

If MuTEAM were provided with a trusted operating system in the sense of the DoD security evaluation criteria at the B or A division level, it could implement the mandatory and discretionary security policies by using security labels on subjects and objects. However, faults can affect the security functions as much as computational functions. If the security functions were made fault-tolerant by using preventive redundancy techniques, the OS could be regarded as trustworthy even in the presence of faults. However, if the software implemented fault-tolerance technique now being used in MuTEAM were extended to also cover the security function, the OS could not be regarded as trustworthy in the event of faults. The prudent action would be then to run the system as if it had an untrusted operating system.

Even if the trustworthiness of the operating system were not in doubt, there would be problems in applying the present MuTEAM's fault-tolerance system to processes and data objects which have different security levels. There is a great deal of message exchange between processes for diagnosis, reconfiguration, and recovery. Violations of the *-property occur whenever two processes at different security level exchange messages, or the receiver process sends some acknowledgement or test result. To handle this, either the fault-tolerance functions would have to be viewed as trusted processes which are permitted to violate the security policy, or fault-tolerance would have to be implemented on process subsets, each at a particular security level. Further, the communications performed for fault-tolerance purposes may permit covert channels which could be used by Trojan Horses in fault-tolerance software. More generally, it may be possible to spoof a node to enter the fault detection and diagnostics mode much more frequently than normal to reduce the system throughput.

With an untrusted operating system, security would be implemented by dedicating specific nodes to single security levels and implementing the security policy with trusted interface units (TIUs). These label all outgoing messages with the node's highet security level. The effect of this is to form subnetworks of nodes, each for a different classification level (omitting any consideration of categories at this time). Since messages with responses cannot be exchanged between subnetworks without violating the security policy or requiring the use of a Guard module, each subnetwork would need to be made fault-tolerant by itself, using the MuTEAM approach. One possible consequence of this is the need of more of the redundant nodes than indicated by the computations to be performed to contain the twin backup processes.

MuTEAM reconfiguration approach is based on an a priori allocation of twin processes to backup nodes and storage of recovery point information at these nodes. However, if dynamic reconfiguration and recovery were introduced, additional problems would arise. For example, if there is no node available with appropriate security level, it would be necessary to prepare such a node either by upgrading or downgrading its nominal security level. In the former case, the other processes in the node would get their outgoing messages labelled higher and, hence, they may not be able to maintain previous communications without using the time consuming Guard process. In the latter case, the higher-classified processes in the node would need to be purged before releasing it to the backup role.

Clearly, imposing a security requirement on a fault-tolerant system such as MuTEAM causes a number of problems which tend to complicate the system or increase its size, or both. Whether or not a suitable solution can be found requires further study.

CONCLUDING REMARKS

We have attempted to establish a framework for determining the impacts of combining security and fault-tolerance in distributed systems, and to set a stage for further research. The preliminary conclusions are that (1) fault-tolerant security requires the use of preventive rather than corrective redundancy, (2) graceful degradation of security is fesible under a particular interpretation, (3) secure fault-tolerance is a complicated problem when software implemented fault-tolerance techniques are used (as in MuTEAM), and (4) more study is requir[illegible].

ACKNOWLEDGEMENTS

This study was supported, in part, by TRW Defense and Space System Group through the Industrial Associates program of the School of Engineering and Computer Science, California State University, Northridge.

REFERENCES

1. Department of Defense Trusted Computer System Evaluation Criteria, CSC-STD-001-83, DoD Computer Security Center, Ft. Meade, MD, 15 August 1983.

2. Computer Security Requirements: Guidance for Applying the DoD Trusted Computer System Evaluation Criteria in Specific Environments, CSC-STD-003-85, National Computer Security Center, Ft. Meade, MD, 25 June 1985.

3. Rennels, D.A., "Fault Tolerant Computing: Concepts and Examples", IEEE Transactions on Computers, December 1984, pp. 1116-29.

4. Siewiorek, D., and R.S. Swarz, The Theory and Practice of Reliable Design, Digital Press, Bedford, MA 1982.

5. Avizienis, A., and J.P.J. Kelly, "Fault-Tolerance by Design Diversity: Concepts and Experiments", Computer, August 1984, pp. 67-80.

6. Habibi, J., Multilevel Security in Distributed Data Processing Architectures with Back-Up and Graceful Degradation, M.S. Thesis, Department of Computer Science, California State University, Northridge, CA, May 1986.

7. Grandoni, F. et al., "The MuTEAM System: General Guidelines", Proceedings of 11th Fault-Tolerant Computing Symposium, June 1981, pp. 15-16.

8. Cioffi, G., et al., "MuTEAM: Architectural Insights of A Distributed Multi-microprocessor System", Proceedings, 11th Fault-Tolerant Computing Symposium, June 1981, pp. 17-19.

9. Bairardi, F. et al., "Mechanisms for A Robust Multiprocessing Environment in the MuTEAM Kernel", Proceedings, 11th Fault-Tolerant Computing Symposium, June 1981, pp. 20-24.

10. Ciompi, P., F. Grandoni, and L. Simoncini, "Distributed Diagnosis in Multiprocessor Systems: The MuTEAM Approach", Proceedings, 11th Fault-Tolerant Computing Symposium, June 1981, pp. 25-29.

11. Barrigazzi, G., A. Ciuffoletti, and L. Stringini, "Reconfiguration Procedure in A Distributed Multiprocessor System", Proceedings, 12th Fault-Tolerant Computing Symposium, June 1982, pp. 73-80.

USER DEFINABLE DOMAINS AS A MECHANISM FOR IMPLEMENTING THE LEAST PRIVILEGE PRINCIPLE

Terry A. Smith
Office of Research and Development
National Computer Security Center
9800 Savage Road, Ft. Meade, MD 20755

ABSTRACT

In the defense and intelligence community, exercising the "need-to-know" principle minimizes the potential damage that can be done by a compromised individual. This is identical to using the least privilege principle in computer systems to minimize the damage that an errant or malicious process can cause. Current security models do not appear to permit the principle of least privilege to be fully implemented. This weakness is exploited by a large class of trojan horses and computer viruses. User definable domains, as developed in this paper, allow the principle of least privilege to be implemented completely, thus providing users with significantly greater protection against these threats.

BACKGROUND

It is an accepted tenet of the government and industrial community that information may be owned. As with any resource, there is a desire to protect information from theft. However, protecting information is very different from protecting other property because information has special properties. Information has value, but who possesses the information directly affects that value. Also, information may be freely duplicated in an undetectable fashion.

One example of this is a football coach who develops a game plan. He desires to communicate this plan, his property, to his players. But only for them to use in a game -- not to tell others. The coach wants to share information yet retain control of its distribution. We term this the information security problem.

This situation also occurs in computer systems. When a user runs code written by someone else, he desires some assurance that this code will not leak information it is given. This is the **confinement problem** as described by Lampson [LAMP 73].

Throughout time, people have struggled with this problem. One highly successful, although incomplete, solution to the information security problem is the military "need to know" system. Under this system each individual is given only that information considered necessary to perform his part of an overall mission. One goal of this system is minimizing the damage that can be done by a compromised person.

The computer analog to need to know is the principle of least privilege. Saltzer and Schroeder identified the least privilege principle as a key design principle for protection mechanisms. [SALT 75] When the least privilege principle is fully implemented each process possesses the minimum access to information required to perform its task.

Current security models implement the least privilege principle by a process having only those privileges explicitly granted by the system. This policy creates what Denning refers to as a **closed environment**. He further states

> The principle of a closed environment is to give each process no more capabilities than it needs to perform its task. The normal state of affairs is completely disjoint, isolated, processes: nothing can be shared or exchanged among processes except by explicit arrangement, all interactions being prohibited unless expressly allowed. No process can attempt to interfere or communicate, with another in an unexpected way. Because a closed environment forces all interactions into the open it is possible to check them all for consistency and validity as desired [DENN 76].

On many computers, it is difficult or impossible to know beforehand what tasks a user will perform. Because of this, users are given a class of privileges which comply with some security policy. <u>e.g.</u> [BELL 75] This often leaves a user process with much more privilege than needed.

This paper proposes that the user be given the ability to further regulate the privileges his processes have. This creates a system of attenuating privileges which allows a user to protect himself from a large class of trojan horse attacks.

it is not proposed that this is revolutionary, but it is believed to be a useful solution to a real problem. In dealing with some security problems, the use of set theory and Venn diagrams is more concise and more direct than conventional matrix notation.

TERMINOLOGY

BASIC SETS

Let X be the set of all users.*
u_i is an element of X.

S be the set of all subjects.
s_j is an element of S.

O be the set of all objects.
o_i is an element of O.

A be the set of all access rights.
a_i is an element of A.

$|X|$ denote the cardinality of X.

*We reserve the use of U for user tuple sets later in this section.

Subjects and Objects may have individual attributes associated with them (e.g. security level, ownership links, etc.).

An **access triple** is defined as a tuple of the form (s_i, o_k, a_m). An access triple is also called a **privilege**. To perform the access described by an access triple is to **exercise the privilege**.

Let M be the set of all access triples.

$$M = (S \times O \times A)$$

At this point we introduce the concept of a **User**.

Definitions:

A User is a human being who is utilizing the resources of a computer system.

A Subject is a process on a computer that performs operations.

A user, upon logging into a system, causes the creation of a process. Through input to this initial process, the user may create other processes and effect changes to data on the system. The concept of these subjects (processes), acting to carry out the wishes of a user, is expressed in a many-to-one mapping of Subjects to Users. This mapping partitions the set S into equivalence classes. If u_i is the image of s_j under this mapping, then we state that s_j is a subject acting in behalf of user u_i. The set of all subjects acting in behalf of user u_i is the **subject set of u_i, denoted K_i.**

We write

$$PHI: S \longrightarrow X$$

where $K_i = \{ s_j \mid PHI(s_j) = u_i \}$.

M may also be partitioned into equivalence classes by Subjects, Objects, and accesses.

A **user tuple set** for a user u_i, denoted U_i, is defined

$$U_i = K_i \times O \times A.$$

This set defines all privileges which may be exercised in behalf of a given user.

SECURITY POLICIES

Suppose that R is a nonempty subset of M. If we designate tuples in R secure and those in not-R nonsecure then R constitutes the instantiation of a Security Policy. A triple in R represents a specific access to a specific object by a specific subject which is considered permissable, hence secure, by the security policy corresponding to R. Thus if M has cardinality m, there are 2^m possible security policies, one corresponding to each of the elements of the power set of M.

COMBINATION OF SECURITY POLICIES

If the intersection of two or more sets, B_i and B_j, is taken, the set produced designates what we call the **combined security policy**. A very important property of this operation is that this combination of security policies may only eliminate tuples from the secure set; combination of a given security policy with others can never compromise (add undesirable access triples to) the initial or any subsequent policy. Thus, freely combining (intersecting) security polices yields a system of attenuating privilege.

TRUST

For a given security policy, R, exercising a privilege deemed non-secure requires **trust with respect to the policy R.**

LEAST PRIVILEGE

For a given task with a given algorithm there exist a minimal set of access triples required to accomplish the task. This is the set of **least privilege** for the task and is denoted **LP[task]**. We assume two different sets of access triples imply two different algorithms.

If **LP[task]** is contained in the subject triple set of subject sa, and is also contained in the secure set of R then the task may be accomplished by sa **without trust.**

DOMAINS

A domain with respect to security policy R has been defined as the list of objects that may be accessed by an entity [SALT 75]. We further constrain the notion by defining a domain as the set of objects that may be accessed and the accesses allowed to those objects. In the set notation a user U_i's domain with respect to a given security policy is defined:

$$D_i = \text{Domain}[u_i] = U_i \text{ INTERSECT } R$$

CURRENT MODELS

Current security models usually separate security into two facets mandatory and discretionary. The mandatory security policy implements restraints required by system owners to protect what they consider sensitive information. The discretionary policy implements restraints chosen by owners (in the computer sense) of information to satisfy their opinions on how the data should be protected. Generally, Access Control Lists (ACL's) are used to implement Discretionary Access Controls (DAC's).

For example, in the Bell and LaPadula (BLP) security model, the mandatory security policy requires that (s_i, o_j, a_k) satisfy the simple security policy and the *-property (read star property). The discretionary policy is satisfied if and only if the access right $\underline{a}$ is in cell (S, O) of the discretionary access matrix. Usually, the accesses listed in the cell are wholly controlled by the owner of the specific object, and are thus ACLs. Therefore, the secure set can be viewed as the intersection of three sets B_1, B_2, and B_3 which correspond to security policies satisfying simple security, *-property, and the access control lists respectively.

More generally, a security policy (sp) embodied by n properties is defined as

$$B_{sp} = \underset{i=1}{\overset{n}{\text{INTERSECT}}} \; B_i$$

where B_i is the set of access triples satisfying the i-th property. Figure 1 shows an example of a BLP security intersection.

THE PROBLEM

EXCESS PRIVILEGE

The partition of M described above may also be viewed from a user's point of view. Intersecting a security policy with a user's tuple set defines the privileges a user may exercise without trust, <u>his domain</u>. Suppose u_i is trying to accomplish task t. Assume that the security policy is B_{sp}. The user's domain is

$$D_i = U_i \text{ INTERSECT } B_{sp}.$$

We assume that the task is accomplishable without trust,

$$LP[t] \quad D_i.$$

This allows the definition of the set of excess privilege,

$$EP[t,i] = D_i - LP[t].$$

AN EXAMPLE

Given a world with three objects of interest, (o_1, o_2, o_3). u_1 owns o_1; u_2 owns o_2 and o_3. The security policy is based on the BLP model. Assume that all files and subjects are operating on the same security level; thus the mandatory security policy is of no concern. u_1 and u_2 give themselves the right to read and write their own files. However, u_2 also gives u_1 the right to write to his file o_2 and to execute the program o_3.

$$\begin{aligned} B_{sp} = (&(s_1, o_1, \underline{r}), (s_1, o_1, \underline{W}), \\ &(s_1, o_2, \underline{W}), (s_2, o_2, \underline{r}), \\ &(s_2, o_2, \underline{W}), (s_1, o_3, \underline{e}) \\ &(s_2, o_3, \underline{e})). \end{aligned}$$

$$K_1 = (s1).$$

$$K_2 = (s2).$$

Suppose u_2 tells u1 that o_3 is a great game. Agreeing, u1 causes the triple $(s_1, o_3, \underline{e})$ to be exercised. The set of least privilege for executing o_3 should be

$$LP[o_3] = ((s_1, o_3, \underline{X})).$$

However the domain of U_1 is

$$\begin{aligned} D_1 = (&(s_1, o_1, \underline{r}), (s_1, o_1, \underline{W}), \\ &(s_1, o_2, \underline{W}), (s_1, o_3, X)). \end{aligned}$$

Thus,

$$\begin{aligned} EP = (&(s_1, o_1, \underline{r}), (s_1, o_1, \underline{W}), \\ &(s_1, o_2, \underline{W})) \end{aligned}$$

In addition to being a great game, the code is a trojan horse: s_1 reads o_1 and writes its contents into o_2 exploiting the excess privilege $(s_1, o_2, \underline{W})$.

Note that (1) an undesired information flow was accomplished, (2) that this did not involve a failure of the security mechanism but exploited a weakness in the security model, and (3) that this flow occurred without the knowledge of the information's owner.

THE PROPOSED SOLUTION

In the above example, eliminating either the tuple (s_1, o_2, $\underline{w}$) or (s_1, o_1, $\underline{r}$) from the set B_{sp}, would prevent the compromise of the file o1. The obvious candidate for removal is (s_1, o_2, $\underline{w}$). However, by the hypothesis that DAC is implemented with ACLs, the only user that can remove it is the owner of o2 which is u2. Present security models appear to grant the owners of objects absolute control over the presence of tuples involving these objects in the discretionary aspect of security policy. One solution to this problem is grant a user control over tuples involving subjects acting in behalf of that user. This control is granted regardless of the object to which a tuple refers.

This is to say that each user u_i defines a subset

$$R(u_i) <= U_i$$

which u_i deems secure. Thus the presence of a tuple

$$(s_j, o_k, \underline{a}) \text{ is an element of } R(u_i)$$

implies that user u_i does not object to s_j, a subject authorized to act in behalf of u_i, to access object o_k in the fashion described by access right $\underline{a}$. This is wholly in the spirit of discretionary access and grants the user powers denied to him by current policies. This subset, a security policy, we term the **User Definable Domain (UDD)** policy for u_i. The set $R(u_i)$ is the domain define by user u_i to implement more completely the least privilege principle for processes acting in his behalf.

Because

$$U_i \text{ INTERSECT } U_j = \{\} \quad \text{for all } i \neq j,$$

we may construct the union of all these sets without permitting users to interfere with each other. This set we term B_{UDD} with

$$B_{UDD} = \underset{i=1}{\overset{|X|}{\text{UNION}}} \; U_i.$$

Now

$$B_{DAC} = B_{ACL} \text{ INTERSECT } B_{UDD}.$$

IMPLEMENTATION NOTES

While it is not the intent of this paper to address implementation questions, a few topics should be addressed to show the feasibility of this model.

COMPUTATIONAL OVERHEAD

The overhead of this scheme could be managed by maintaining a master security matrix S x O x A. This ties the overhead of updating security sets to the frequency with which users update their domains and ACLs -- this frequency can be regulated by policy according to each system's needs. An additional advantage of this scheme is that the burden of updating these tables can be charged directly against the users initiating the update.

Of course completely maintaining such a matrix is not practical on real systems. The size of the matrix would overwhelm the computing power of some machines and burden the rest. However the most users would have sweeping domains where all members of a certain group are excluded. The matrix could be factored along these lines and only important subsets of the matrix maintained dynamically.

SECURING REQUIRED TABLES

The information user use for domain definition needs to be protected. However, this is identical to the problem of securing ACLs, and whatever device used to secure ACLs should be able to protect UDD information.

USER INTERFACE

The model, as stated, assumes the user has a fair amount of knowledge about security and operating systems. Since this is not the case usually, both the implementation and administration of a security model should address the problems of securing the security-ignorant and/or computer-ignorant user.

For instance, a system could have default domains for the user, user groups and system functions. Then the system might initially set up user accessible commands (via trusted path) <trust> and <distrust>. The <trust> command would add tuples to R(u); the <distrust> command would remove them. These utilities would allow sophisticated commands in the following fashion:

```
$trust write to user=bruce
(individual)
$distrust read to group=VLSI_DESIGN
(group)

$trust read to directory=HOME
(location)
```

User defaults would be set up by sophisticated commands to protect against standard attacks yet allow normal functioning.

```
$trust file_creation to directory=HOME and -
                     owner=me and
                     acl = none

$trust write to user=ME and directory=HOME or
-                    directory=SYSTEM_TEMP_DIR
```

This family of commands might also include hooks into the DAC system so that a user could say "write only to files which are also not readable by the verification group."

VERIFICATION ISSUES

In a verified system that maintains a master security matrix, we believe the UDD system can be retrofitted to provide some level of assurance by verifying two things. As a scenario, a user u_j describes to a program the tuples he wishes to remove from the matrix. This program generates Ø, a subset of M. Next a process requests the operating system to remove Ø from the master security matrix, R. The system verifies that this request originated with a subject acting in behalf of u_j and does the following computation (using something akin to Pascal)

P := P INTERSECT U_j ;

This assures that the user is removing only his own tuples. Next the system updates the master security matrix by computing

R := R - P ;

At a minimum, the operations of validating the identity of the user and the set operations described above must be verified correct and then made tamperproof. For true security though, the interface which initially generated the set P must also undergo verification.

AN EXTENSION

Often, an individual will use a computer system in two different roles. For example in a small software design team, one person might be both a programmer and an archivist for group code. The domains applicable to these two tasks are likely to be very different.

To account for this need, users could also have more than one domain, R(u,k). At process creation time the user could specify the domain to be used. For example in a VMS system the command could be,

```
$SPAWN/NOWAIT/SECURITY_DOMAIN=personnel RUN
payroll
```

or in Unix (if we must)

```
%payroll -sd personnel &
```

CONCLUSION

We are defining security policy as the intersection sets. We enhance current security models by including a user-definable domain. These domains allow a user to constrain information flows initiated by subjects acting in his behalf. We have for every user u_i, the triple (s_j, o_j, $\underline{a}$), with s_j is an element of K_i, is secure with respect to u if and only if

(s_i, o_j, $\underline{a}$) is an element of B_{MAC} INTERSECT B_{DAC} INTERSECT B_{UDD}

where B_{MAC} is the set of triples satisfying the mandatory access constraints of the initial model, B_{ACL} the sets of triples satisfying discretionary access constraints, and B_{UDD} is the set of those satisfying additional conditions devised by the users to closely approximate the least privilege set required for their processes.

The idea of protection mechanism that are subject oriented instead of object oriented is not a new one[DENN 82]. It is believed that this technique can be used to enhance existing models in an efficient (as compared to model's initial computational overhead) manner, that the additional proofs required for verified systems will be manageable, and that it will provide a high level of assurance against discretionary trojan horses (including computer viruses).

ACKNOWLEDGEMENTS

The author would like to express his thanks to Will Harkness for invaluable assistance in the development and preparation of this paper.

BIBLIOGRAPHY

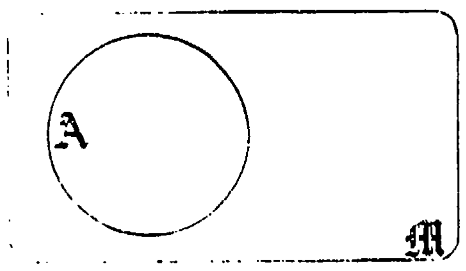
LAMP 73. Lampson, Butler W., "A Note on the Confinement Problem," Communications of the ACM V 16 # 10, October 1973.

BELL 75. Bell, D. E. and L. J. LaPadula, Computer Security Model: Unified Exposition and Multics Interpretation, tech. report ESD-TR-75-306, AD A023588, The Mitre Corporations, Bedford, MA, June 1975.

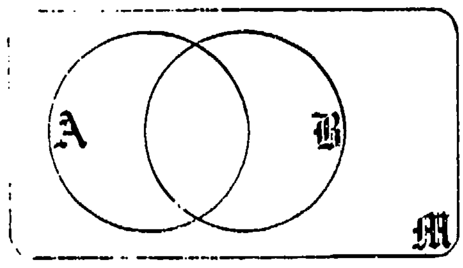
DENN 82. Denning, Dorothy E. R., Cryptography and Data Security, Addison-Wesley Publishing Company, Reading, MA, 1982.

DENN 76. Denning, Peter J., "Fault Tolerant Operating Systems," Computing Surveys, Vol 8 No. 4 December 1976.

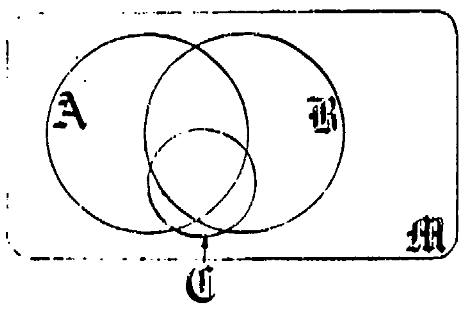
SALT 75. Saltzer, Jerome H. and Schroeder, Michael D. "The Protection of Information in Computer Systems," Proc. of the IEEE, Vol 63 No 9 September 1975.



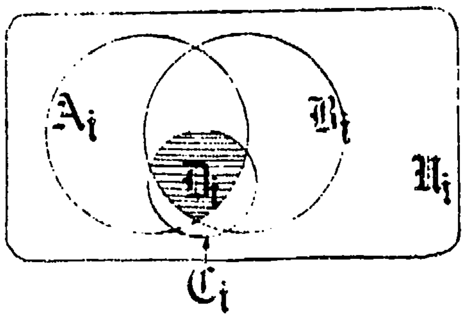
A = Tuples not eliminated by *-property.



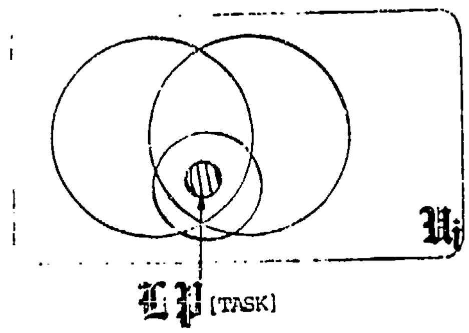
B = Those not eliminated by the Simple Security Property.



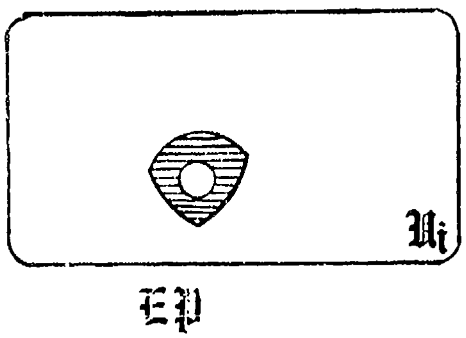
C = Those not eliminated by Discretionary Controls (ACL's).



In the User Tuple Set, this defines that user's DOMAIN.



LP = The set of Least Privilege for some task.



The Set of Excess Privilege .

FIGURE 1

A Venn Diagram Illustration of the Concepts Being Discussed. (Note: in the fourth figure the plane set switches from M to U_i.)

INTRODUCTION TO THE ACCESS PATH

TECHNICAL DEVELOPMENTS

Telecommunications networks, dispersed processing capabilities, increased storage capacity and the introduction of software components with advanced functions have totally changed the mainframe computer environment. When data processing was centralized, the computer and all terminals were located in an area where they could be directly controlled. Concerns about the data and the computer were often about control over physical access to the computer room itself. Computer operators initiated all processing and loaded all programs and data files as needed. Output was reviewed by operations and/or data control personnel who ensured all jobs were processed to completion and the results appeared to be correct. They then released the output to the user departments. Many of these systems were relatively simple-the user could often review and reperform the computer generated reports to determine the correctness of processing.

On-line real-time data entry with immediate update of transactions to data files has removed the necessity for much of the batch processing. The current level of sophistication totally removes the operator from much of the day-to-day control over the processing. Most data files and programs are permanently resident on the computer. Operators cannot identify each transaction that is processed, nor determine its appropriateness or its effect on each data file. The possible lack of hard copy audit trails could mean the user has little or no chance to retrospectly review processing in its entirety.

The risks associated with data processing have increased as both the speed with which information can be changed and the number of users accessing the system have rapidly increased. Telecommunications has vastly expanded the number of terminals linked into the mainframe and the degree to which they are used. The introduction of minicomputers and microcomputers, linked both to each other and to mainframes, has distributed processing even further. These new users may have little appreciation for the risks of data loss, error or fraud.

The traditional control features such as physical access and division of duties can potentially be compromised in these sophisticated systems by any person having access to either a terminal or another computer linked to the mainframe. Management, users and data processing staff all need to place significant reliance on the appropriate functioning of the computer network to reduce the risk of incorrect or unauthorized processing.

USING THE ACCESS PATH

People unfamiliar with large computers or people familiar with traditional batch systems may find that identifying potential risks and controls in the maze of software components which make up sophisticated networked computer systems to be overwhelming. A simple but useful method of gaining and recording an understanding of these systems is to prepare an Access Path diagram.

An Access Path diagram is a concise one-page depiction of all the software components in the system under review, and depicts the sequence of information flow from one component to the next. It is a very useful instrument when recording, explaining or discussing a system, especially for identifying the risks and controls which may be present.

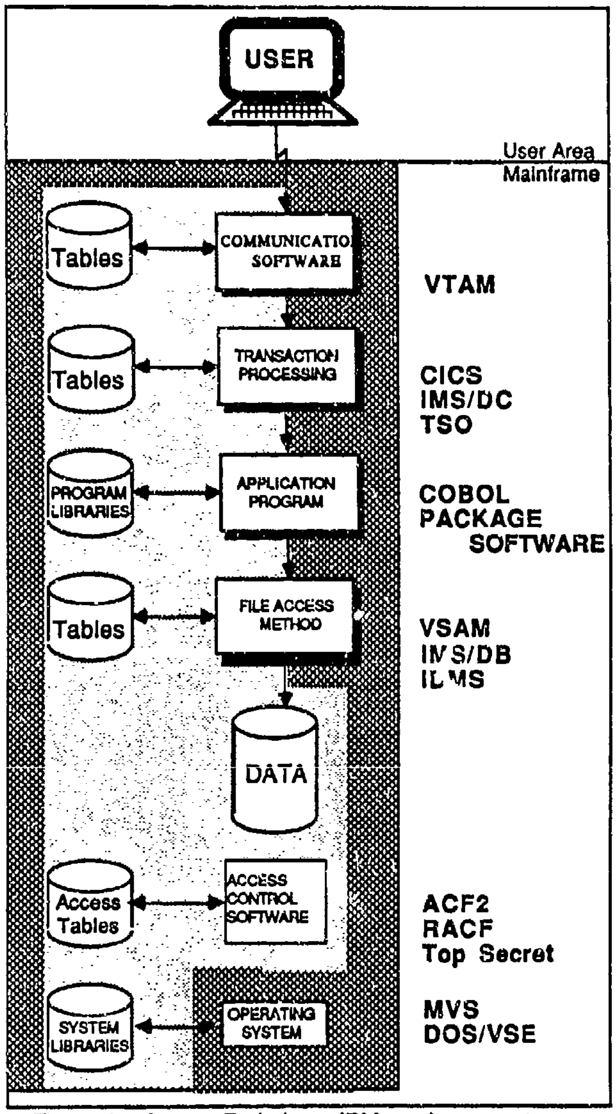


Figure 1: Access Path in an IBM environnment

RISKS AND CONTROL CONSIDERATIONS

Built into these multiple layers of software are features that may affect the risk and control considerations such as passwords, user identification codes, transaction codes, access level indicators and processing options. The same software components can be implemented differently in every location. For example, most access security software packages provide various options for the treatment of unauthorized access attempt: it could disallow the access and not report the violation; it could disallow the access and report the violation for investigation; it could allow the access and report the violation; it could allow the access and display a warning message to the user; it could allow the access and ignore the violation completely.

How these features within each component are installed and maintained significantly affects the risks and controls present in each system. The Access Path shows a broad view of the software components involved and where the installation may have used the control opportunities available in each component. If an in depth evaluation of the system is required, more detailed information can then be obtained for those software components identified as being relevant.

DESCRIPTION OF THE SOFTWARE COMPONENTS

Figure 2 illustrates the view that most users have of their data access - an uninterrupted direct connection. They generally do not understand, and should not have to understand, all the software involved in accessing the data. This section will give a brief description of the actual activities that take place transparently each time a user - for example, an accounts receivable clerk - accesses data in a typical mainframe computer installation.

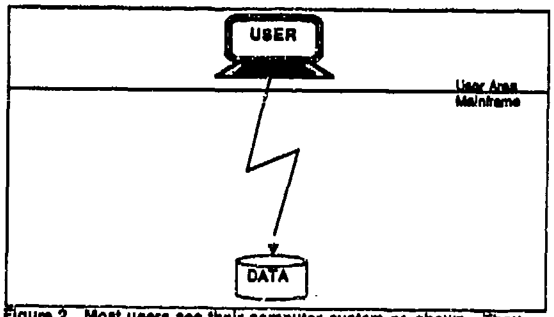


Figure 2. Most users see their computer system as shown. They know that there are terminals and files involved, they may not realize how the files are accessed.

TELECOMMUNICATIONS

The first component of system software encountered on the access path is the Telecommunications Software (illustrated in Figure 3). This software connects all the terminals, printers and other peripherals to the rest of the computer system. All of these links need to be defined within the telecommunication software if the rest of the system is to receive messages from or submit messages to any of these peripherals.

The software continuously "listens" to all the terminals for any messages being transmitted. When it "hears" a message, it identifies the terminal, retrieves the message and transmits it to the system. Likewise, it "hears" and retrieves messages from the system and transmits them to the terminals indicated.

The telecommunications software used by most IBM mainframes is the Virtual Telecommunications Access Method (VTAM). VTAM is made up of several different programs which essentially perform the following functions:

* Recognize that a terminal is attempting to submit a message

* Identify the terminal and check if it is defined within the telecommunication software

* Ensure the access authority, as defined within the telecommunication software, is appropriate for the intended message

* Package and transmit the message to the next component of the access path

* Recognize that a message is being sent from the system to a peripheral and route it accordingly.

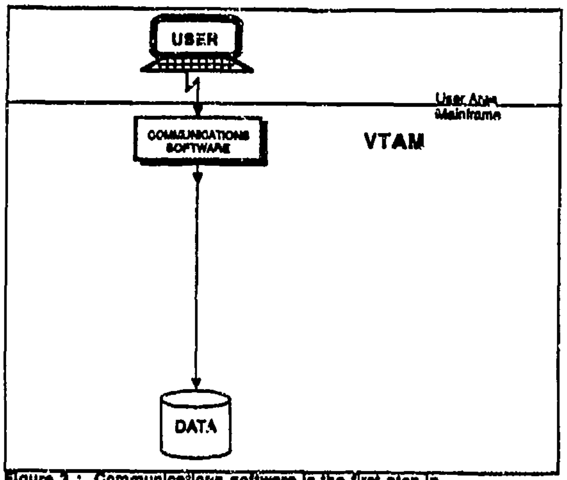


Figure 3 : Communications software is the first step in the access model. For the system to accept a message from a terminal, it must be defined to the computer via this software. At this point, the communications software provides the computer with the message being sent as well as the identification (terminal) from which the request is coming. Communications software can limit the functions that can be performed by the terminal. In addition, there may be password protection available.

TRANSACTION PROCESSING SOFTWARE

The second component of system software on the access path is the Transaction Processing (TP) software (illustrated in Figure 4). TP software is used by many computer installations to specify which terminal can use application programs (exceptions to this rule are the Time Sharing Option (TSO) and other text editors. These are discussed below.

TP software serves as the link between the telecommunication software and the application program which actually processes the message, and can be used with both batch and real-time systems. The message transmitted by a user contains a name or code whereby the TP software can identify its nature. The TP software then performs various control functions which enable the message to pass along to the required application program for processing if the access is permitted by the access tables contained within the TP software.

The most commonly used Transaction Processing software on IBM mainframes is Customer Information Control System (CICS). Information Management System/Data Communications (IMS/DC) is an alternative for a system which uses an IMS data-base management system. CICS can perform many functions, but broadly described it performs the following:

* Handles all terminal messages transmitted to and from the Telecommunication Software (described in the previous step)

* Schedules the execution of all processing activity within CICS. Each component of processing activity is called a task

* Controls the information flow to accommodate multitasking. This allows more than one task to be submitted at a time, although only one task will be executing at any specific point in time

* Controls the loading and releasing (unloading) of the application programs required to execute the tasks.

The Timesharing Option TSO) and other text editors (WYLBUR, ROSCOE, et al) were designed primarily as productivity aids for application and system programmers. Use of these editors does not tie the terminal to specific programs as does CICS and other TP software. Rather, the editors make utility programs available which allow programmers to read files, tables and libraries; scan them; change them and/or delete them. In addition, the editors allow submission batch jobs that are handled by the system like any production job.

Figure 5 shows the power that editors such as TSO give the programming staff. In essence, all files, programs and tables are potentially available to the user of such an editor.

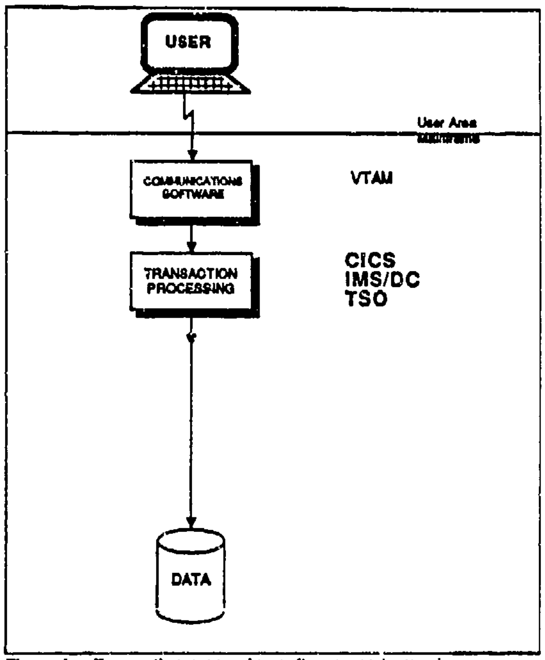


Figure 4 : Transaction processing software can be used to limit access to the system by matching specific transactions to the terminals and/or users. Limited password security measures can also be implemented here.

Log files can be produced that can be used for backup/recovery purposes as well as to determine (audit) system usage.

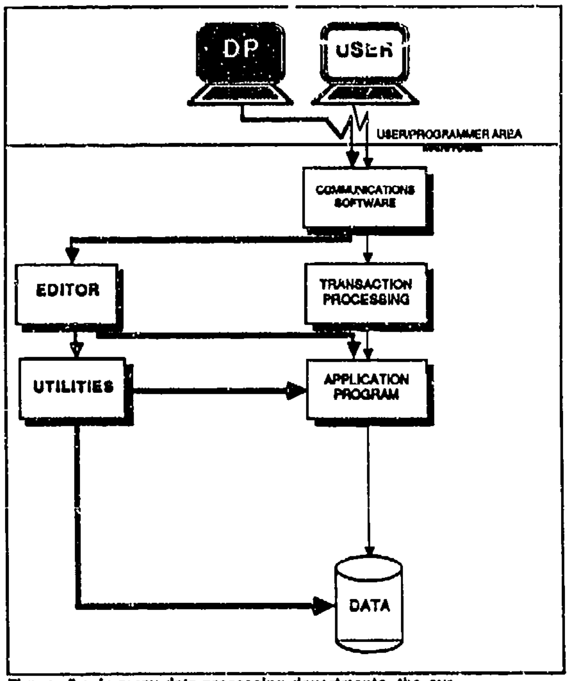


Figure 5 ; In many data processing departments, the systems and applications programmers have access to utilities that allow them to add, delete and/or change programs and data files. Due to the capabilities of the utilities, close supervision and review of their use may be necessary. Additionally, care should be taken relating to any "user" activity that programmers are allowed to perform.

APPLICATION PROGRAMS

After the TP software has identified the nature of the message and completed the necessary control functions, it transmits the message to the relevant application program. This is the program that will perform the actual function required by the user, for example: do a calculation, search for the available quantity of an inventory item, or print an invoice. Whereas one Telecommunication Software package and one Transaction Processing Software package normally handle all the information flowing within a given access path, there may be tens, hundreds and even thousands of application programs which can be used within the same access path.

Many organizations are now purchasing and installing application programs for common business systems which have been developed by independent software vendors rather than employing their own programmers to develop the applications. This purchased software is also referred to as "packaged" or "off-the-shelf" software, and although it is most often associated with microcomputers there are numerous vendors selling application software for mainframes. Figure 6 illustrates the Application Program in the Access Path.

Application programs are written in a variety of programming languages. The majority of business application programs are written in COBOL.

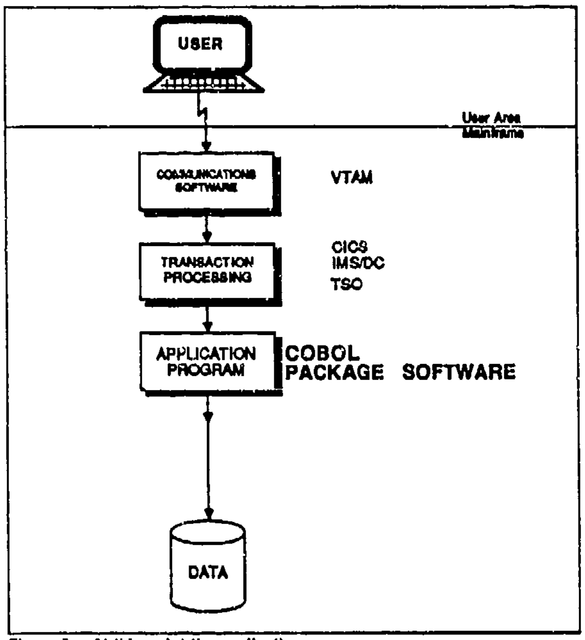


Figure 6 : At this point the application programs are accessed. The program analyzes the data received to determine how the transaction should be processed and recorded on the files. Editing, reasonableness checks, etc. can be performed here. In addition, the program may read and write to files, format and send messages to the originating terminal and can perform additional security related functions.

Many application software packages can be purchased from third-party vendors. The user of purchased software should ensure that the software meets their needs.

FILE ACCESS METHOD

Almost all application programs require the manipulation of data in some manner. Data may be read, added, deleted or changed. Data is stored in files which may be on magnetic disk or tape. Although the application programs issue the instructions directing the manipulation, it is the File Access software that actually retrieves the data from and writes the data to the files. In data processing terminology, these manipulations are referred to as the input/output (I/O) operations.

There are many different file access methods. Two of the most commonly used are Indexed Sequential Access Method (ISAM) and Virtual Storage Access Method (VSAM). Figure 7 illustrates the File Access method in the Access Path.

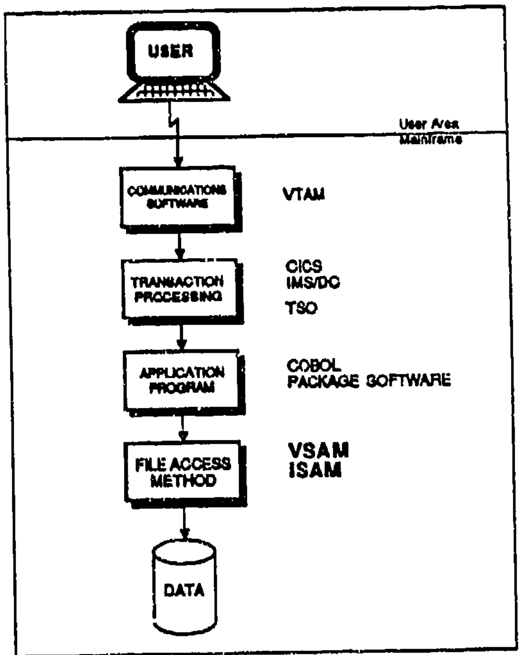


Figure 7 : The file access method function relates the requesting program to the files needed for processing and the method by which the data will be stored. In other words, it actually performs the input / output operations.

Where an installation uses a data base rather than conventional computer files for storing data, there is an additional component in the access path. The data storage and organization is controlled by a Data Base Management System (DBMS). Two of the most frequently used data base management systems on IBM mainframes are Information Management System (IMS) and Integrated Data Base Management System (IDMS). Any data access has to pass through the DBMS to obtain the exact storage location of such data. Within the DBMS a File Access method performs the actual input/output operations. These components are illustrated in Figure 8.

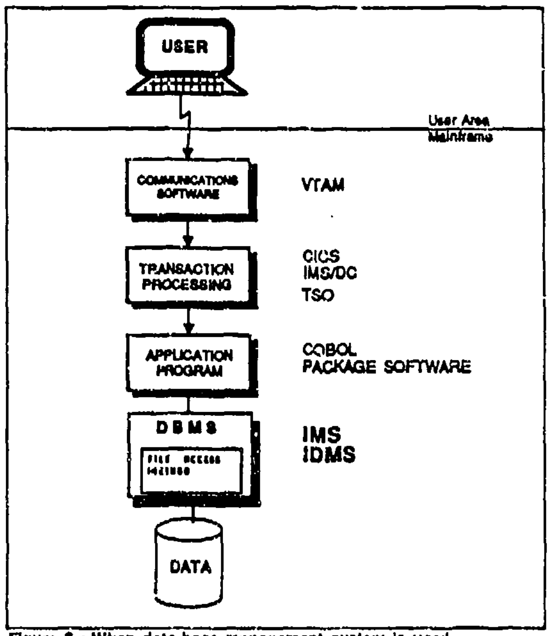


Figure 8 : When data base management system is used, it normally includes its own file access method to perform the I/O operations to and from the database.

OPERATING SYSTEM

The Operating System is an integrated set of programs that control and coordinate the operation of the computer. It interacts with all the previously mentioned steps in the Access Path and allows all the components to communicate with each other. Part of the system software, it is a set of programs that directs the computer system. It can translate high-level languages (e.g., COBOL) into machine language (with a compiler), manage system resources (tape and disk files, program libraries, etc.), retrieve information from files, schedule and supervise work, and operate and control mechanized devices (tape and disk drives, computer terminals, etc.). It is not specific to any one application, but may be used in the design, processing and control of all applications and other system software components.

The Operating System provides the operators control over starting up and shutting down the computer, and controls the allocation of resources to enable the computer to process efficiently and handle multiple users accessing the system at the same time. Figure 9 shows the operating system as part of the Access Path. All activity within the shaded area occurs under the control of the operating system.

There are two basic operating systems used on IBM mainframes. The earliest system, introduced in the 1960's, is the Disk Operating System (known as DOS/VSE). The later system, introduced in the mid-1970's, is Multiple Virtual Storage (known as OS/MVS). Although the two systems perform similar functions, there are many differences between them, and switching a computer from one system to the other requires a major effort.

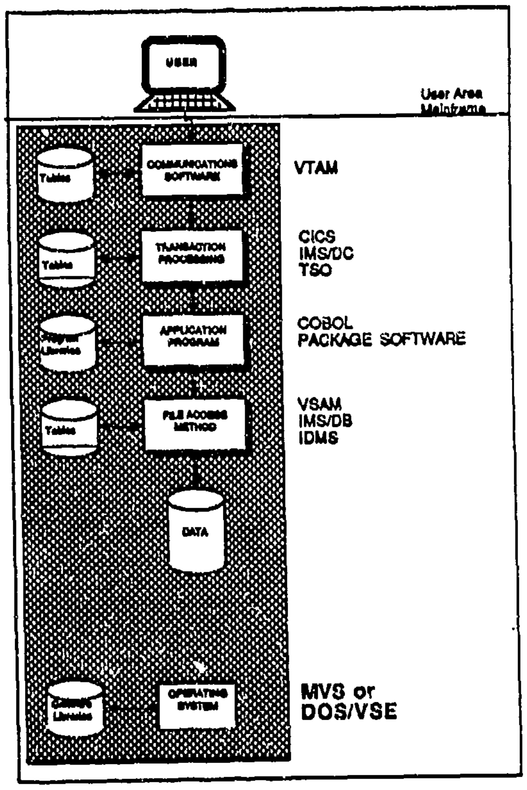


Figure 9. The operating system software has control over all the previously mentioned "steps". Since there can be many users on the system at the same time, "traffic control" is necessary to give processing time to each task. The operating system schedules each task to ensure that each user is given appropriate priority and the correct resources for the job (files, disk drives, etc.).

ACCESS CONTROL SOFTWARE

As computer systems have become more sophisticated, the number of users, transactions and software components have increased. In order to limit the access each user has within the system a number of Access Control software packages have been developed. These packages are designed to protect the data files, program files and system software files within the installation considered to be vulnerable. All accesses permitted within the system are defined in access tables, and the system then compares every action attempted against these tables to determine if the action will be permitted. Obviously these systems are only effective if the access rules defined in the tables are functionally appropriate, correctly implemented and accurately maintained. Figure 10 illustrates the access control software operating within the access path.

The three most frequently used access control packages on IBM mainframes are RACF, (an IBM developed package) and two independently developed packages, ACF2 and Top Secret.

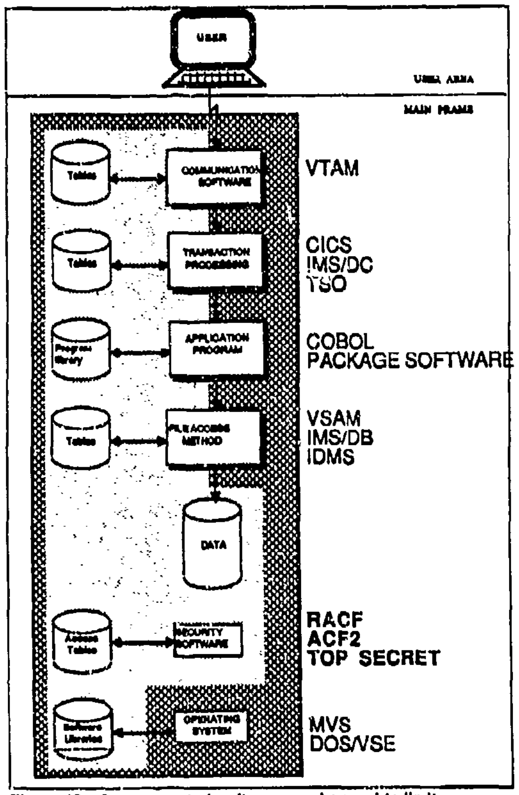


Figure 10. Access control software can be used to limit access to files, libraries and tables held on the computer. The proper implementation of such software can aid in providing a secure system. On-going monitoring of the system is required to ensure that the security policies and procedures are followed.

THE AUDIT AND REVIEW OF THE ACCESS PATHS

The access path is a methodology developed to help the auditor define and evaluate system security and other restrictions against unauthorized access in a complex environment. It is an easy way to understand system and application software interaction. An auditor needs to know: who can access what data files? The Access Path provides him a way of identifying every possible way of accessing one file. Each way is a different access path.

Thus, the first step in the audit or review process is to identify the files or data elements which are of interest to the auditor.

Every access path to these sensitive files should then be mapped. Most installations have several access paths to the data files. The path can be the batch processing of production jobs, on-line access by the user department, use of utility programs by application programmers, etc. Every path will be made up of different kinds of software. Access to data is controlled by these different layers of system software and application programs. At each layer, there may be controls that prevent system users from performing tasks outside of management's intentions (see Figure 11). How these controls are actually employed is an audit concern.

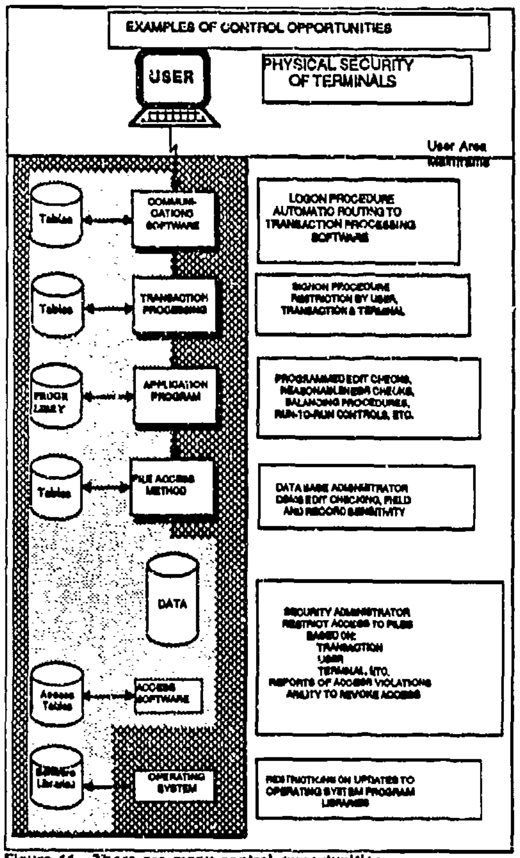

EXAMPLES OF CONTROL OPPORTUNITIES
USER
PHYSICAL SECURITY OF TERMINALS
User Area
Mainframe
Tables
COMMUNICATIONS SOFTWARE
LOGON PROCEDURE AUTOMATIC ROUTING TO TRANSACTION PROCESSING SOFTWARE
Tables
TRANSACTION PROCESSING
SIGNON PROCEDURE RESTRICTION BY USER, TRANSACTION & TERMINAL
PROGR LIBRARY
APPLICATION PROGRAM
PROGRAMMED EDIT CHECKS, REASONABLENESS CHECKS, BALANCING PROCEDURES, RUN-TO-RUN CONTROLS, ETC.
Tables
FILE ACCESS METHOD
DATA BASE ADMINISTRATOR DBMS EDIT CHECKING, FIELD AND RECORD SENSITIVITY
DATA
Access Tables
ACCESS SOFTWARE
SECURITY ADMINISTRATOR RESTRICT ACCESS TO FILES BASED ON: TRANSACTION USER TERMINAL, ETC. REPORTS OF ACCESS VIOLATIONS ABILITY TO REVOKE ACCESS
Software Libraries
OPERATING SYSTEM
RESTRICTIONS ON UPDATES TO OPERATING SYSTEM PROGRAM LIBRARIES

Figure 11. There are many control opportunities available to the personnel responsible for the design of the system and computer network. The auditor should understand the technology as implemented within the data center, recognize the methodology being used and incorporate it into the audit plan.

The auditor will review as part of the audit, the security features for each software mapped on a path giving an access to any sensitive data files. The auditor, or reviewer, knowing the control opportunities which each software product offers, will record, for all these software products, which of those opportunities have been implemented.

For example, the security key feature of CICS might be employed to restrict access to CICS transactions. The result of these inquiries provide a map of the Access Path, with the implemented controls which restrict access and therefore indicate the areas which merit testing. The testing, in this case, will involve the review of the user profiles and tables which the relevant software products reference in order to restrict access. This review will typically have to be conducted with the use of software for the purposes of printing out the profiles or tables. This software may be a feature of the product itself, a general purpose utility program which is supplied by the vendor or a software product expressly designed for this purpose. Coopers & Lybrand has developed, and continues to develop, customized software for this purpose (i.e. the CICS Analyzer). Having reviewed those profiles for the users and reached an opinion as to their adequacy, the next step is to consider and review the paths which can be taken by other categories of user. It is also important to remember that the tables and profiles which have been reviewed above will be the subject of maintenance. This is because the community of users is normally constantly changing as a result of employees joining and leaving the company as well as transfers from one department to another. Also, it should be remembered that the application programs and system software products are also being changed. Therefore, it is important for the access paths to the tables and profiles be mapped and a review be conducted of the adequacy of the controls over this change management process.

In order for this review to be complete, it is important to consider all categories of user who potentially might have access to the data and programs and profiles or tables. In the steps above, we have reviewed the end-user and those who are responsible for maintaining the profiles and tables. It is also necessary to review the access paths that the members of the Data Processing Department use. Obviously, when reviewing the Access Paths that are mentioned above, the main purpose is to ensure that only authorized users have access to authorized facilities. One detailed aspect of this would be to ensure that members of the Data processing Department do not have access to production versions data files using production versions of programs. Members of the programming section of a data processing department will typically have access, via the system, to various operating system utility programs, typically via an editor such as TSO, for example. They will also have the ability to submit batch jobs for processing. It is with these programming tools that they could gain access to line or production versions of programs and data files, thereby bypassing controls which are contained not only in the application programs but also in the system software components. It should be remembered that these "bypasses" must exist in most installations for valid operational reasons. For example, the database may require repairing after the computer went down because of a power failure or a logic error in a program. The essential consideration here, though, is to ensure that the use of these access paths is suitably restricted and authorized. This review will involve using software to print out reports showing the user profiles in the editor (e.g. TSO) and considering the appropriateness of the entitlements given to each user.

After having identified the active security features within all software layers of every possible path to sensitive data files, the auditor is now in a position to evaluate the risk of unauthorized access to these files.

RISK ANALYSIS AND COMPUTER SECURITY: BRIDGING THE CULTURAL GAPS

Lance J. Hoffman
Department of Electrical Engineering and Computer Sciences
The George Washington University
Washington, D. C. 20052
(202) 676-4955

Abstract

Specific problems which currently limit the effectiveness of computer security risk analysis are discussed. These problems have already surfaced and in some cases been addressed by the risk analysis community outside of computer security. It appears that the quality of computer security risk analysis can be significantly improved by using previous work or undertaking certain basic steps in these areas.

1. INTRODUCTION

In recent years, significant changes have taken place in the computer security field. With the explosive growth of personal computing, computer system penetration from home is now a reality which is constantly demonstrated [Park83]. In response, port protection security devices have been developed. New products have come to market (and continue to) in other areas of computer security as well; over a hundred were exhibited at one recent trade show.

Work in traditional areas of computer security research (e.g., authentication methods, cryptography, statistical inference protection) continues [Proc86], especially research into the development of trusted operating systems for multilevel secure operation (spurred on by trusted system evaluation criteria [NCSC83]). However, with the increased realization that mo-r computer security problems and solutions are not entirely technical, and with the increasing number of real-world systems at risk, the risk analysis process has lately received additional attention [Cecu86, Guar85, Hoff85].

While risk analysis is an interdisciplinary area, computer security specialists have in general not used models and techniques from other fields to the extent possible. There has been some cultural gaps between the risk analysis community which has been developing models and techniques for risk assessment and risk management and the computer security community which has until recently largely concentrated on either technical or administrative solutions without paying a great deal of attention to exposure assessment, risk characterization (including uncertainty), or weighing of alternative solutions.

This is not surprising since risk analysis (like computer security) is a multidisciplinary field requiring a blend of skills; the development of any such field must cross disciplinary boundaries and breach semantic barriers. Few in the computer security community know of existing results in probabilistic risk analysis or even of the existence of the Society for Risk Analysis or its journal; similarly, very few risk analysts have paid any attention to the computer security literature. And this type of work, like any which crosses jurisdictional boundaries, has trouble attracting support from traditional sponsoring organizations or universities.

Unlike traditional risk analyses which deal with concete consequences (such as money or lives lost), often computer security concerns are diffuse and intangible (e.g., military advantage, competitive advantage, privacy protection). Asset values are more difficult to arrive at, since data can be an ambiguous asset whose value varies significantly over time and by use. And perhaps more so than in other areas, operational priorities put real-world constraints on what computer security measures will be used. Still, the existing risk analysis literature may be able to shed light on the costs of breached security.

In the past, the computer security community has often used ill-fitting adaptations of risk analysis methods that were developed for problems which were significantly different. The majority of computer security risk analyses have used annual loss expectancies (ALEs), a method well-suited to and used by insurance companies. However, unlike insurance risks, computer security risks often are multiple and not readily specified (by money, injury, or death). Often the potential losses are intangible--related to national defense, corporate goodwill, or other nonmonetary assets. Unlike traditional risk analysis problems, computer security problems tend to often lie in a relatively uncharted area, that of diffuse risks from adversarial sources, where the objects at risk and the nature of the risk may be diffuse and where the source of the risk may be a malevolent adversary. These risks might be characterized as points on the right of Figure 1 [Brow86].

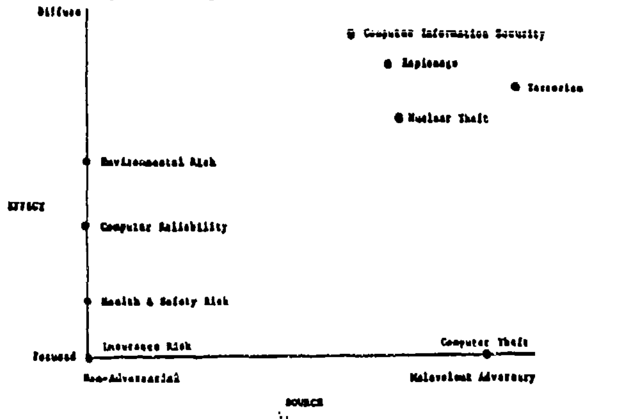


Figure 1. Categorization of Risk Analysis Problems by Type and Source of Risk (Brow86)

2. MAJOR PROBLEMS IN COMPUTER SECURITY RISK ANALYSIS

There are several areas where significant problems exist which currently limit the effectiveness of computer security risk analysis. The problems appear to be tractable; by bringing resources to bear on them, the quality of computer security risk analysis can be significantly improved. This section describes the specific areas and suggests appropriate actions to take.

2.1. Semantic Problems Due to a Lack of Standard Definitions

A critical area where research is needed is that of standard definitions in risk analysis for computer security. While there are accepted terms in both fields, sometimes the same term means two different things, depending on the field; in some cases, there are differences among workers even in computer security; in a few cases, there are out and out conflicts between the commonly accepted definitions in risk analysis and usage in computer security. By and large however, these conflicts appear resolvable if addressed promptly; both fields are relatively new and the leaders appear quite willing to work together to agree on one common set of terms.
There is a computer security glossary produced by the National Bureau of Standards which contains several hundred definitions which has been out for several years; a more recent one is [NCSC85] from the National Computer Security Center. Even so, in the workshop which led to this paper [Hoff86], "we had significant trouble with communication among computer security people" [Cour85]. There does not appear to be any widely used formal glossary of terms for the risk analysis field in general. It is absolutely necessary that the two disciplines communicate well; therefore, harmonization of existing definitions is needed and a common glossary of terms would be helpful.

2.2. Absence of Guidelines on When to Use Risk Analysis

Another important issue is when to use risk analysis. Too often in the past, computer security practitioners have either avoided initiating a risk analysis due in part to fear of its cost or, alternatively, initiated full-fledged analyses which slavishly used methodologies better suited for other problems and, as a result, cost more and produced less of value than desirable and possible. Apparently no guidelines exist regarding when a risk analysis or a specific methodology should be initiated or terminated.

It is inappropriate to use (certain kinds of) risk analysis when the potential benefit gained from such use is too small. A related issue is how far to go; one may often need a relatively simple or even cursory analysis and nothing more. Alternatively, one may begin an analysis, suspend it for a time, and then resume it as events warrant. Finally, in some cases a full-blown exhaustive analysis may be called for.

As an example, often safeguards which cost the least displace the most risk; organizational policy statements and employee awareness programs can be relatively easy to cost-justify, and may in many cases obviate the need for more detailed risk analyses. But the problem is not always that simple, and in particular safeguard selection can be complex:

> ..."A baseline look might, at times, let you identify generic measures which should be taken - but you cannot implement generics; you must implement specifics. To identify the specific measures needed you need to look in far greater detail than is normally considered in some initial, cursory inspection and understand the need for a set of fully complementary measures." [Cour85]

2.3. Communicating risk management options to decision makers

Public perception of computer security breaches often involves "hackers" dialing in from afar to obtain protected information or to "crash" a system [Levy84, Psyc80]. However, the reality is that outsiders are much less likely to cause computer problems than are data errors and omissions, dishonest or disgruntled employees, a failure of administrative controls, or water damage. This is often not communicated effectively by computer security professionals to their management. We thus have the real-world problem of the risk analysis that, once done, sits unread upon a shelf. This is often the case even when the results are appropriate and accurate and the analysis was done efficiently. Typically this happens because top management was not convinced of the need to take any corrective action. Often, management reacts to events rather than planning protective measures in advance. What may appear reasonable to a security manager may be excessive when looked at through the eyes of a higher level decision maker.

Indeed, one of the most vexing problems risk analysts and computer security experts have is communicating risk management options to decision makers. Risks and adverse events are often not popular topics, and the options available may all be undesirable. Presenters of risk management options have to avoid a number of pitfalls. On the one hand, they may be accused of being too analytical and cost-oriented and insensitive to human or political costs which are difficult to quantify; on the other hand, addressing those important issues but not having enough credible data on which to base a decision lays them open to charges of being vague. Insensitivity to either of these can spell doom to any hope of selecting reasonable options even if excellent data is in hand (which is never the case). Ignoring interdependencies may also lead to unrealistic risk assessments and thus, when discovered, cripple the credibility of an analysis. Misapplication of automated tools and unwarranted belief in their output is another potential problem. And of course no methodology will be useful in an institution that doesn't want to know what the risks are; it takes an organizational commitment to make the results at all useful [MacG86].

Disciplines which are, at the first glance, far removed from scientific risk analysis -- advertising, communications, psychology, etc. -- might contribute to better communicating risk management information and choices, especially since there so may opportunities for misinterpreation of results as diverse audiences are addressed. The area is so new that the first major national conference on communicating risk to the public took place in January 1986 (the National Conference on Risk Communication, Mayflower Hotel, Washington, D. C., January 29-31, 1986, sponsored by The Conservation Foundation, National Science Foundation, Environmental Protection Agency, American Industrial Health Council, and the University of Southern California). Efforts to improve this situation may go farther than anything else to mitigate, in the long run, the real problems risk analysts are asked to address.

2.4. Lack of Test Beds and Respected, Available Studies

There is a critical lack of test beds and of well-known, respected impartial risk analyses of computer system security to use as examples. It is difficult to evaluate the performance of various risk analysis methodologies or tools without a suitably rich test bed. With one or two exceptions, such a research asset does not exist and the fields' growth and maturity is hindered by incomplete testing and information.

In test bed development, as in real world risk analyses, there is very little case data available on which to base estimates or assessments; and computer security personnel have long bemoaned the fact that estimates of threats are hard to elicit and very hard to justify; there is not enough historical data. Thus, a few test beds and pilot studies which incorporated traditional risk analysis techniques with real problems from computer security (and other application areas) and using real data would be very important in advancing research progress by helping us to develop, based on real world experience, a general model and conceptual framework for computer security risk analysis. Notions of generalization, methodological development, and demonstration should be in mind, while at the same time carefully focusing the efforts. The scope must be narrow enough to be manageable; one would hope that at the end the result could be a highly visible successful application of known techniques and models to a real computer security problem. After that, other efforts can be held up to that standard.

2.5. Problems with available data

In any undertaking such as test bed development, data base problems will be run into. The most likely of these is lack of data. Risk analysis and computer security experts have long bemoaned the fact that there is very little real-world case data available on which to base estimates or assessments. Real world case data collection would help matters, both in the general risk analysis case and in the specific computer security case. Examples of such data are the relative frequencies of different types of security breach and the measurable impact on security of specific incidents and of various risk management measures.

Elicitation of this information is not easy, and relatively highly trained individuals must be available to encode the data for later use; this task cannot be left to unknowledgable persons. Worthwhile also would be an effort to review existing data and data gathering efforts related to computer security such as data banks maintained by Donn Parker at SRI International; Robert Courtney of Robert Courtney, Inc., and Glenn M. Jones of the Pentagon Joint Data Services Support Center. After such a review, significant gaps would be identified and the process of gathering new needed data could be started. Such work will require knowledgable persons to encode the data. An initial effort at this is underway at the National Computer Security Center, under the direction of Roy Wood.

2.6 Uncertainty

Estimates of threat likelihoods are hard to elicit and validate; nevertheless, the risk analysis community has already made some important progress in the area of uncertainty by using probability distributions to quantify uncertainty about exposures and severity of effects. In particular, work in nuclear safety by Rasmussen [NRC75], and in the more general field of probabilistic risk assessment [Howa76, Morg84, Henr85, Cox81] is relevant. However, this Bayesian, probabilistic approach is only a start, and there remain quite a few unanswered questions related to uncertainty. There is, for example, a growing literature on low probability, high loss events. Nevertheless, we are still uncomfortable handling these in the real world.

A number of important questions with respect to uncertainty remain unanswered in the specific area of computer security [Henr86]:

> Should all risks be quantified? Should all uncertainties about numerically expressed risks be quantified? Are linguistic expressions of severity and uncertainty ("rarely", "likely") sufficient, or are they inevitably bedevilled by ambiguities? If not, are probabilities always the best approach? What of fuzzy sets, Dempster-Shafter calculus, and various other approaches to representing uncertainty, both quantitative and qualitative, developed by researchers in artificial intelligence and expert systems? One fuzzy set based approach [Schmucker] has

already been applied to computer security. Is it better than the others?

Studies which compare these approaches (on both theoretical and practical criteria), assess their merits and drawbacks, and start to develop guidelines about which may be appropriate under what conditions are needed.

2.7. Desirability of a general risk model as a conceptual framework

If a general risk model could be developed which could be used by both the risk analysis and computer security communities, it would provide significant benefits to both communities in the areas of testing, methodology evaluation, and completeness of analysis. Such a conceptual framework should be very flexible, be able to handle numerous types of risk analysis computer security problems, and be able to handle all external policies imposed on the problem. It should be able to be easily refined as new knowledge (for example, from the test beds and pilot studies described above) becomes available or new constraints appear. It should be acceptable to both communities and rich enough to represent just about all computer security situations. Without such a model, "we will continue to have methods that are as different as apples, oranges and pears and which will produce results which cannot be compared" [Katz85].

The interrelationships between threats, threat frequencies, vulnerabilities, safeguards, risk, outcomes, etc., should all be described in a formal way so that a common understanding of the risk analysis process emerges [Katz85]. Such a model might, at least in part, not be highly mathematical, since it would ideally make effective use of case-based and quasi-statistical data described in Section 2.5. It should be able to handle but not be limited to techniques such as the FIPS PUB 65 annual loss expectancy method [NBS79]; commercial methodologies such as return on investment method [Coln85] or Bayesian decision support [Ozie86]; and perhaps even a qualitative fuzzy set theoretical approach [Schm84].

The model would provide generic threats, assets, etc. as well and would fit a number of specific methods described in [Hoff86]. It must also allow for approximations to those functions we do not know how to define. This will allow us to implement tools, in the near term, that represent simplifications to more complex interrelationships. As we obtain more insight about interrelationships, we should be able to replace the simplistic representations with more complex ones. Furthermore, it must allow alternative methods for different purposes; it should allow appropriate combination of qualitative and quantitative input data; and it should be consistent when applied to the same problem by two different teams of people using the same data.

Finally, it should be a "living model" (in the words of H. O. Lubbes) which is able to constantly change, as the life cycle of the system goes on, and reflect the updated configuration of the system.

2.8. Dearth of Metrics for Risks and for Risk Analysis Methodologies

One significant lack today is metrics for risk analysis and risk management. There is no currently accepted set of criteria against which all methods can be compared. It is difficult evaluate or to convey the advantages and disadvantages of a given methodology or tool when no accepted evaluation metric exists. Until such a set of criteria is developed, we can expect proliferation of various methodologies, most of which are adaptations of previous ones (even if the previous ones have serious deficiencies).

A deeper problem is the lack of metrics for risk, even within the risk analysis community. There has been little research on value tradeoffs to guide policy decisions (with some notable exceptions such as the roughly $1 million value put on a human life in airline safety risk analysis, and $1,000 cost equivalence of a man-rem of exposure used in nuclear regulation). In computer security such metrics (e.g., a dollar value put on a breach of secret defense information) are scarce. One such is an initial attempt at a multi-attribute utility function related to congressional options on a number of issues in information security [Brow85]. There is also little work on generalization of binary logic to multiple states which handle reliability with degraded performance (e.g., a safeguard that works some of the time or which partially works.

2.9. Appropriateness of Automation

Recently, there has been a proliferation of computer security risk analysis tools and products [Hoff85, Fiks85, Henr85] which are particularly useful in getting the risk analysis started, allowing quick sensitivity analyses, and producing reports. Despite these advantages, the risk analysis community has been quick to caution against premature development or use of quantification, automation, or expert systems. They are concerned that "Issues of modeling, uncertainty assessment and judgment of value require the kinds of thinking that software tools can't provide right now" and should only be used for routine calculations, such as implementing the logic of fault trees.

In the workshop which led to this paper, all agreed that there should not be a rush to computerize and that automated tools should not claim or imply more than is there; in essence, automated tools are fine as a means, not as an end. Some subtle dangers are involved here also, including the lack of credibility when the systems don't deliver what they promise and the locking in of inappropriate methodologies by premature computerization and inflexible software.

No one was willing to advocate the idea of using expert systems or artificial intelligence in risk analysis today, at the early stage of development these fields are in. However, if a suitable general model can be built, then prior experience in the codification and treatment of expert opinion might be used in the development of an expert system to produce a risk management tool which would be quite useful. This would be a lot more than an electronic checklist: it would, based upon information related to the specific installation being analyzed, suggest actions to take to improve security. Such systems have been built or proposed in many areas, including medicine and business planning. Naturally, all the rules used by such a system would have to be traceable and clear, and the system would have to handle nontechnical as well as technical risk to computer systems.

3. ACKNOWLEDGMENTS

The workshop which led to many of these conclusions was supported by contract MDA904-85-C-A478 to The George Washington University from the National Computer Security Center of the National Security Agency of the Department of Defense at Ft. Meade, Maryland. Dr. Sylvan Pinsky was the technical manager for the Center. His initiative and willingness to examine an interdisciplinary area with potential high payoff but about which relatively little is known is recognized and appreciated. This paper is extracted from the complete report [Hoff86].

4. REFERENCES

(Brow85) Brown, R. V., Presenting risk management information to policymakers: Executive summary of a report to the National Science Foundation. Technical Report 85-4, Decision Science Consortium, Inc., Falls Church VA, July 1985.

(Brow86) Brown, R. V., "Managing Diffuse Risks from Adversarial Sources (DR/AS) with Special Reference to Computer Security: Ideas for a New Risk Analysis Research Area", Working paper 86-1, January 1986, Decision Science Consortium, Inc., Falls Church, VA 22043.

(Cecu85) Cecula, d., "Consider alternatives to formal risk analysis", Government Computer News, September 27, 1985.

(Coln85) Basic Data Systems, Inc., The Risk Analysis Machine, Rockville, MD, 1985.

(Cour85) Courtney, R. L., Private communication, September 24, 1985

(Cox81) Cox, D. C. and Baybutt, P., "Methods for Uncertainty Analysis: a Comparative Survey", Risk Analysis, Vol. 1, 1981, 251-258.

(Fiks85) Fiksel, Joseph, "Automated Threat Assessment for Computer Facilities", A. D. Little, Inc., Cambridge, Mass. 02140, 1985.

(Guar85) Guarro, S. B., Garcia, A. A., Wood, C. C., and Prassinos, P. G., LRAM: Livermore Risk Analysis Methodology for Information Systems Security, UCAR-10150, December 1985, Lawrence Livermore National Laboratory, Livermore, CA.

(Henr85) Henrion, Max and Morgan, M. Granger, "A Computer Aid for Risk and Other Policy Analyses", Risk Analysis, Vol. 5, No. 3 (Sept. 1985), 195-208.

(Henr86) Henrion, Max, Private communication, January 30, 1986.

(Hoff85) Hoffman, Lance J., "PC Software for Risk Analysis Proves Effective", Government Computer News, Vol. 4, No. 18, September 27, 1985, pp. 58-59.

(Hoff86) Hoffman, Lance J., Computer Security Risk Analysis: Problems and Issues, Report GWU-IIST-86-04, Department of Electrical Engineering and Computer Science, The George Washington University, Washington, D. C., March 1986.

(Howa76) Howard, R. A., Matheson, J. E., and Miller, K. L. (eds.), Readings in Decision Analysis, Decision Analysis Group, Stanford Research Institute, Menlo Park, CA, 1976.

(Katz85) Katzke, Stuart, "Summary of Key Issues", in (USAF85).

(Levy84) Levy, Steven, Hackers: Heroes of the Computer Revolution (Garden City, New York, Anchor Press/Doubleday, 1984)

(MacG86) MacGregor, D., Private communication, January 14, 1986

(Morg84) Morgan, M. G., Morris, S. C., Henrion, M., Amaral, D., and Rish, W. R., "Technical Uncertainty in Quantitative Policy Analysis--a Sulfur Air Pollution Example", Risk Analysis, Vol. 4, No. 3 (1984).

(NBS79) National Bureau of Standards, Guidelines for Automatic Data Processing Risk Analysis, FIPS PUB 65, Gaithersburg, Md., August 1979.

(NCSC83) DOD Computer Security Center, Department of Defense Trusted Computer System Evaluation Criteria, CSC-STD-001-83, 15 August 1983.

(NCSC85) National Computer Security Center, COMPUSECese Computer Security Glossary, NCSC-WA-001-85, Ft. Meade, Md., October 1985.

(NRC75) NUREG-75/014, Reactor Safety Study, an Assessment of Accident Risks in United States Commercial Nuclear Power Plants, WASH-1400 Study, Nuclear Regulatory Commission, Washington, DC, October 1975.

(Ozie86) Ozier, Perry, and Associates, Bayesian Decision Support System, San Francisco, 1986.

(Park83) Parker, D. B., Fighting Computer Crime, Chas. Scribner's Sons, New York, N. Y., 1983.

(Proc86) Proceedings of the 1986 IEEE Symposium on Security and Privacy, Computer Society Press, Catalog No. 86CH2292-1, Oakland, CA.

(Psyc80) "The Hacker Papers", Psychology Today, Vol. 14 (Aug. 1980), p. 67.

(Schm84) Schmucker, Kurt J., Fuzzy Sets, Natural Language Computations, and Risk Analysis, Computer Science Press, Rockville, MD, 1984.

(USAF85) Minutes of the Federal Information Systems Risk Analysis Workshop, 22-24 January 1985, Air Force Computer Security Program Office, Gunter AFS, AL [available through Defense Technical Information Center, Alexandria, VA].

MANAGING DIFFUSE RISKS FROM ADVERSARIAL SOURCES (DR/AS) WITH SPECIAL REFERENCE TO COMPUTER SECURITY

Dr. Rex V. Brown
Decision Science Consortium, Inc.
7700 Leesburg Pike, Suite 421
Falls Church, Virginia 22043
(703) 790-0510

ABSTRACT

An essentially new methodological area of risk analysis is proposed, in which the risks are multiple and diffuse and the source of risk is a human adversary. Computer security is a special case of particular interest. The methodological needs for both risk assessment and risk management, dealing with these types of risk, are defined and related to the current state-of-the-art in other branches of risk analysis and decision analysis. Distinctive analytic techniques are suggested, extending the existing armory of analytic tools for risk analysis. Issues and approaches include: formulation of risk consequences (e.g., macro models and plural analysis); evaluating risk consequences (e.g., via multiattribute utility functions and alternative devices); predicting adversarial behavior (game theory and decision analytic models); predicting complex risk aftermaths (step-through simulation); determining institutional and social value; specifying the impact of action options; and choice and implementation of options.

1. INTRODUCTION

1.1 Evolution of Risk Analysis Methodology

Risk analysis, as a distinct field of inquiry, has been steadily evolving in terms of the complexity of risks it addresses, and thus requires increasingly ambitious analytic tools. Risk situations might be characterized along two dimensions: the source of the risk and the effect of the risk. The source might be represented along a continuum between non-adversarial and a malevolent adversary. The effect of the risk might range along a continuum between focused (or exact) and diffuse (or inexact). These dimensions of risk analysis are shown conceptually in Figure 1.

1.1.1 Simple focused risk, non-adversarial source. Historically, risk analysis research first addressed the simplest type of risk: focused risk whose source is nature--i.e., non-adversarial. This risk is typified by the risk faced by insurance companies. Risk is focused in that it can be expressed along a single, easily measured dimension, such as money; and the bearer of the risk is a single entity, such as a corporation, a joint venture, or an individual. In some of these cases, such as life and health insurance, risk assessment is simplified by the availability of substantial historical records, which permit uncontroversial determination of probabilities. In other cases, human judgment must play a significant role in the assessment, typified by a recent case where Lloyd's of London insured against the discovery of the Loch Ness Monster (for a manufacturer of scotch who had offered a million pound reward to the happy discoverer). However, this still remains the simplest case of risk analysis.

1.1.2 Multiple focused risk, non-adversarial source. Within the last ten years, risk analysis research has expanded to consider more complex risks, ones that are somewhat more diffuse than the simple risks that are addressed by insurance companies. This risk is typified by health or safety risks of the type addressed by many governmental regulations (see Figure 1). It is somewhat more diffuse than the first case, because the objects at risk are multiple and the risks themselves are multiple, even though the risks are defined along easily measured dimensions (such as death and health effects) and the objects at risk are easily specified (such as human populations). The source of this risk, however, is non-adversarial, and is often the combination of nature and technology (such as drugs or nuclear power plants). As with life insurance, quantification of the risks can typically be anchored to observed frequencies, but human judgment has a significant role to play (for example, in predicting events that have never occurred, such as a reactor core meltdown).

This health and safety area has now achieved some considerable measure of technical maturity in predicting, evaluating and managing the risks involved, with significant support, for example, from the Risk Analysis Program at NSF. It uses, among other techniques, personalized (Bayesian) probability for quantifying risks, and multiattribute utility theory (MAUT) for trading of death against disability against economic cost. The literature in this area is now quite extensive, and the state-of-the-art is reasonably represented in the following selected references: Covello & Menkes[1]; Keeney & Raiffa[2]; Lave[3]; Ricci, et al.[4]; Rowe[5]; Risk Analysis[6]; Schwing & Albers[7].

1.1.3 Diffuse risk, non-adversarial source. A risk which is substantially more diffuse, but still non-adversarial (i.e. technological) is typified by environmental risk analysis, where multiple ill-defined effects are experienced by often equally ill-defined objects at risk (see Figure 1). Environmental risk analysis has been spurred during the past decade or so by the National Environmental Protection Act (NEPA), the establishment of the Environmental Protection Agency, and the resulting requirement for environmental assessments and environmental impact statements for a wide range of projects (Leape[8]). Like most health and safety risk analysis, it is largely motivated by government regulation.

1.2 New Risk Analysis Requirements

New categories of risk are now emerging, which require a new analytic technology, which can accommodate risk analyses in the whole plane of Figure 1. That is, where the objects at risk and the nature of the risk could be diffuse, and in where the source of risk could be a malevolent adversary. Such risks might be termed "diffuse risks from adversarial sources" (DR/AS). Most types of computer security are prime examples of this type of risk, to be discussed below. Other examples include theft and sabotage at nuclear and other energy facilities, espionage and terrorism in its many forms. These risks might be characterized as the points in the top right corner of Figure 1. Multiple effects are not necessarily diffuse. One of the multiple effects of environmental risk might be the destruction of a wildlife sanctuary, which is quite focused, compared with the breached security effect of a computer system, which can only be evaluated with consideration of a possibly complex pattern of "aftermaths" leading, for example, to possibly harmful uses of information by potential enemies of the United States.

The existing analytic methodology is not well adapted to the new levels of complexity introduced by this class of risk. There have been isolated instances of promising methodological innovation, developed in the process of solving specific practical problems, for example, consequence evaluation for nuclear safeguards. However, little has been done to unify or generalize them. Systematic approaches have also been developed on analogous problems (for example, modeling adversaries in negotiation and competitive situations, and modeling complex future scenarios in military planning). However, they have not been adapted for, or applied to, the problem of analyzing and managing risk.

Although nearly all of recent risk analysis literature has been on physiological risks from technological sources plus a little on environmental risks and on natural hazard sources (as typified by the coverage of the journal Risk Analysis), DR/AS risk analysis has not been entirely lacking. It has, however, been piecemeal and typically case-specific. At Decision Science Consortium, Inc. (DSC), for example, we have performed risk analyses for nuclear safeguards against theft, malevolent acts against energy facilities, and international monitoring of nuclear proliferation, and methodological innovations have been developed and applied. However, such developments have not been systematized or codified for general use.

Analytic techniques for modeling diffuse future effects are being developed through application areas other than risk analysis, notably defense planning, which needs to take into account the unfolding of complex military scenarios. Various forms of scenario specification and simulation have been devised including step-through simulation (Ulvila, Brown, & Randall[9]; Ulvila & Brown[10]), which economizes on mental burden. These methods are, on the whole, at an early stage of development and have not been adapted to problems of DR/AS.

A distinctive aspect of "adversarial source" of risk is the role of motivation and perception, which interacts in complex ways with risk management efforts. For example, where there are several alternative ways for realizing a hazard, (e.g., breaches of information security in a computer system, or ways for a proliferator to divert nuclear material), a risk manager's success in blocking one path, if perceived by the adversary, may lead the latter to reassign his effort in other directions. Again, analytic approaches to this class of problem have been attempted in non-risk fields, notably game theory (Luce & Raiffa[11]; Shubik[12]), and the use of prescriptive decision analysis models to predict adversarial and other human behavior. For example, Brown et al.[13] uses prescriptive decision analysis models to predict NATO response to an impending Warsaw Pact attack.

Interactive decision theory, which incorporates concepts from both decision analysis and game theory, has been developed for negotiation applications, and also has suggestive analogies with the case of DR/AS risk analysis (Raiffa[14]; Ulvila[15]).

1.3 Computer Security as a Special Case

Most computer security risks are special cases of this new area, but some types (e.g., computer theft) have focused effects (money), and others have non-adversarial sources (e.g., computer reliability).

The tools currently available for risk analysis within the computer security community draw very little on previous risk analysis work, partly because the computer security community is generally unaware of this work and partly because computer security problems, being largely DR/AS, have not always been readily amenable to many of the traditional risk analysis techniques. The computer security community has, in the main, been using ill-fitting adaptations of risk analysis methods that were developed for significantly different problems. One of these is the Annual Loss Expectancy (ALE) method (FIPS PUB 65) based on practices in the insurance business, where the risk of concern is that of losing money in insurance claims. Usually these methods are not appropriate in situations where the cause of the risk is a human adversary and where the effects of the threat are diffuse. With computer security, there may be a human adversary, such as a "hacker" (Levy[16]) and the effects of the threat are diffuse because once data is compromised, it may be impossible to precisely specify the effects.

In computer security, multiple risks must be considered, and these risks are not always easily quantified (as contrasted with money, injuries, or deaths). The threats will vary from one installation to another. The countermeasures available to handle the threats include not only technical measures, but also physical and administrative security techniques. As in many other areas, there are very little case data available.

2. RESEARCH NEEDED

Specific analytic techniques need to be developed to address the distinctive features of risks which are multiple and diffuse and the source of risk may be a malevolent adversary. Computer security would be an excellent special case to exercise them on.

Developing an appropriate methodology for DR/AS problems can build on past work, such as the state-of-the-art of risk analysis as used in conventional application areas (Risk Analysis[6]), case studies, completed and ongo ing, of specific attempts to analyze computer security and other DR/AS problems (Brown[17]); a review of decision science methodology for problems analogous to DR/AS (Brown, et al.[13]); and initial efforts to develop a methodological paradigm for the new dimensions (Brown & Lindley[18]; Ulvila & Brown[10]; Brown & Feuerwerger[19]).

In keeping with standard risk analysis practice, we distinguish two analytic tasks: risk assessment and risk management. Risk assessment involves quantifying the probability of unfavorable outcomes in the absence of any deliberate intervention. Risk management involves the evaluation of potential measures to manage the risk, i.e., to reduce it or its consequences. Both phases involve identifying potential relevant consequences, their probabilities of occurrence, and their evaluation if they do occur.

An appropriate unifying methodological perspective is that of personalized decision analysis, which incorporates human judgment in quantifying uncertainty and value in the process of prescribing action (Raiffa[20]; Brown, et al.[21]). A schematic outline of one such analysis is given in Figure 2.

For this new type of DR/AS problem, we suggest that its methodological needs for both risk assessment and risk management need to be defined and related to the current state-of-the-art in other branches of risk analysis and decision analysis. Within this new area, we propose a research plan for developing methodologies where the needs are greatest and with special application to of computer security. The methodology can be exercised on live cases, primarily in the process of conducting a complete risk analysis for an unclassified version of a live problem of computer security at a large defence facility. We now describe this plan more fully.

3. DELINEATING THE NEW TYPE OF RISK ANALYSIS (DR/AS)

To set the stage for a broader program of methodological and data gathering research we argue that DR/AS is a class of risks (typified by large areas of computer security, nuclear safeguards, espionage and terrorism) that is distinguished from the more conventional areas of risk analysis in similar ways, such that they could be usefully studied together, leading to the development of a unified methodology. The dominant distinguishing features shared by this class of problem refer to the nature of the risks and their sources as they bear on methods for assessing them and evaluating their seriousness.

The features are the diffuseness of the risks (including multiple dimensions, multiple and ill-defined risks, and uncertain consequences extending over time); and the fact that the major source of risk is a human adversary (whose behavior is not susceptible to the same prediction methods as inanimate or at least nonmalevolent sources). There may be other dimensions of analogy or disanalogy to be explored.

As shown in Figure 1, not all risk analysis problem areas fall cleanly in or out of the DR/AS category. For example, theft of proprietary data stored in a computer system ("Computer theft" in Figure 1), as a subcategory of computer security, has the element of an adversarial source (e.g., the thief), but the risk itself may be monetary (and to this extent has much in common with the financial risk facing insurance companies). Conversely, there are areas where the risk is diffuse (such as certain kinds of environmental impact), but the source is inanimate and technological and in this respect similar to health and safety risk analysis (see Figure 1).

Special attention needs to be paid to the distinctive methodological needs of computer security as a legitimate area of risk analysis research in its own right. A DOD-sponsored workshop on computer security risk analysis, chaired by Lance Hoffman of George Washington University[22], was recently held. It was suggested there that a general conceptual model for computer security can be developed and used to model unauthorized disclosure, destruction, modification of data and denials of service from the point of view of risk analysis. The problem of setting a value on intangibles deserves to be examined, as well as problems involved in characteriz ing and propagating uncertainties (Brown[23]) which are to date almost unrecognized by the computer security community. Also of interest are problems in communicating computer security risks to various risk management actors (e.g., Congress and facility managers) and constituency groups (e.g., segments of the general public, system manufacturers and vendors, end-users, government administrators).

4. DEVELOPING SELECTED METHODOLOGIES FOR DR/AS PROBLEMS

Specific pieces of new methodology need to be developed to address the most serious deficiencies in the current state-of-the-art applied to DR/AS risk analysis problems, in the light of the results of the effort described in Section 3. They can be illustrated in the context of computer security and other DR/AS examples, and address both of the standard divisions of risk analysis: risk assessment and risk management.

4.1 Risk Formulation

We are concerned with risky situations where the possible consequences are diffuse, i.e., they cannot readily be characterized by a few simply specified standard events or measures, such as monetary costs, a core-melt accident, or a number of deaths. The question: "What is risk?" cannot be simply formulated in operational terms and, indeed, the appropriate formulation may vary with the situation and defy standardized definition.

Alternative methods and principles for formulating risk deserve investigation. At this time, it appears that four possibilities show a high degree of promise. First, risk might be specified in a "macro model" containing few high-level, abstract attributes that could be expected to span the range of concerns in given risk assessment. For example, computer security risk might be specified along such attributes as national security, economics, privacy, cost, civil liberties, and others. As another example, a macro model of the risk of nuclear material theft (shown in Figure 3) might specify risk along the attributes of material stolen, deaths, damage, possession of material by adversary at any time, apprehension of adversary, penetration of safeguard system.

•Second, the macro model might be extended to represent the risk from the points of view of several different constituencies. For example, the risk at a U.S. government computer facility might be represented from the point of view of the facility manager, the U.S. legislature, and several segments of the public. As another example, the risk of nuclear material theft might be represented from the points of view of facility manager, government regulator, and society, as illustrated schematically in Figure 2.

Third, since macro models may be too highly aggregated and broad to represent all important details of the risk, a series of "feeder" models may be developed and incorporated into the modeling process to provide detailed analyses of the most important aspects of risk. For example, a macro model of nuclear power plant risk might use a feeder probabilistic risk assessment (PRA) to provide a detailed analysis of the magnitude of accidental exposure to radiation.

Fourth, since no single specification of a diffuse risk is always adequate for all pur poses, techniques of "plural analysis" (Brown and Lindley[18]) should be investigated. Plural analysis involves pursuing two or more separate approaches to the same problem and then formally reconciling or pooling the different results.

4.2 Consequence Evaluation

In addition to problems of formulating risks, situations with diffuse risks pose problems in evaluating the consequences of the risk. The problem is due primarily to the need to characterize multiple attributes of risk. Thus, not only do the individual attributes need to be assessed, but comparisons across different categories of risk must be determined. For example, in computer security risk, total risk might be characterized in terms of national security, economics, privacy, cost, and civil liberties, and others. As assessment of total risk requires a method to compare a level of risk on one attribute with the level of risk on another attribute. Multiattribute utility analysis (Keeney and Raiffa[2]) offers a promising method for development such comparisons.

It is worth exploring general ways, both of defining appropriate scales for multiattribute utility analysis, and of deriving value parameters that compare different attributes. The methods can be exercised in the context of computer security. A tenta tive example is presented in Brown[17], which evaluates alternative national computer security policies. One might also build on other related work which involved developing an index of hazard for radioactive waste, which is reported in Watson[24]. This involved field work to elicit value judgments from three sources: members of the general public, the responsible Government administrator, and technical experts. The analysis can be either weakly or strongly quantified (see Figures 5 & 6, respectively).

4.3 Modeling the Aftermath of a Risk Event

An alternative method to macro models for handling diffuse risk is Monte Carlo simulation, where complex possible consequences of aftermaths to a risk are represented as a sampling of possible complete paths. However, for diffuse risks, the conventional Monte Carlo simulation requires specifying probabilities for all possible contingencies and poses an unmanageably heavy burden. We have developed an alternative, called "step-through simulation," in the context of diffuse consequences of military actions, where an expert and a model interact in producing each trial (Ulvila, Brown, and Randall[9]; Ulvila & Brown[10]).

4.4 Modeling Adversarial Behavior

The methodological significance of the "adversarial source" of risks is that anticipation of deliberate, hostile human action requires special assessment techniques, which are not appropriate for inanimate, or at least nonadversarial, sources of risks (c.f. human error in the operation of nuclear plant). A major avenue to be explored is modeling the decision processes of the adversary.

Game theory (Luce and Raiffa[11]; Shubik[12]) is one implementation of this idea, though the need for restrictive assumptions on the rationality of behavior and extensive information available to the adversary severely limit the practicality of this approach.

A more promising alternative is to anchor prediction of an adversary's behavior to that which a decision-analytic model of his choice would indicate. This has been used in a study concerned with probabilistically pre dicting a NATO response to an impending War saw Pact attack (Brown et al.[13]). The literature for predicting deliberate, but nonadversarial, human action can also be reviewed for applicability.

A key element in this prescriptive approach to prediction, which needs substantial development, is handling the slippage between prescription and prediction, acknowledging the fact that the adversary may not behave as the decision analysis of his choice would in dicate. The psychological work of Duncan Luce[25] and the interactive decision analysis work of Howard Raiffa[14] provides a starting point. There has also been some more specific work on this problem, in the context of predicting nuclear theft behavior of malevolent acts against energy facilities (Hill[26]).

5. CONCLUSION

The object this paper has been only to get out, for comment and suggestion, some preliminary ideas on what might constitute a fruitful new area of risk analysis. We believe it will call for distinctive--and major--research and methodology development, on a scale comparable to that which has been devoted in recent years to health and safety risk analysis.

REFERENCES

1. Covello, V.T., & Menkes, J. Issues in risk analysis. Hohenemser, C., and Kasperson, J.X. (Eds.). Risk in the technological society. AAAS Selected Symposium 65. Boulder, CO: Westview Press, 1982, 287-301.

2. Keeney, R.L., & Raiffa, H. Decisions with multiple objectives: Preferences and value tradeoffs. New York: Wiley, 1976.

3. Lave, L.B. (Ed.). Quantitative risk assessment in regulation. Washington, DC: Brookings Institution, 1982.

4. Ricci, P., Sagan, L., & Whipple, C. (Eds.). Technological risk assessment. Alphen an den Rijn, The Netherlands: Sijthoff and Noordhoff, 1983.

5. Rowe, W.D. An anatomy of risk. NY: Wiley, 1977.

6. Risk Analysis, Special issue on nuclear probabilistic risk analysis. Vesely, W.E. (Guest Ed.), December 1984, 4(4).

7. Schwing, R.C., & Albers, W.A (Eds.), Societal risk assessment. NY: Plenum Press, 1980, 181-216.

8. Leape, J.P. Quantitative risk assessment in regulation of environmental carcinogens. Harvard Environmental Law Review, 1980, 4, 86.

9. Ulvila, J.W., Brown, R.V., & Randall, L.S. Step-through simulation: A method for implementing decision analysis (Technical Report 76-18). McLean, VA: Decisions and Designs, Inc., November 1976. (NTIS No. AD A036969).

10. Ulvila, J.W., & Brown, R.V. Step-through simulation. Omega: The International Journal of Management Science, 1978, 6(1), 25-31.

11. Luce, R.D., & Raiffa, H. Games and decisions. New York: Wiley, 1957.

12. Shubik, M. Game theory in the social sciences. Camgridge, MA: M.I.T. Press, 1982.

13. Brown, R.V., Kelly, C.W., III, Stewart, R.R., & Ulvila, J.W. A decision-theoretic approach to predicting the timeliness of NATO response to an impending attach (U). Journal of Defense Research, May 1977, Special Issue 77-1 (Crisis Management), 126-135.

14. Raiffa, H. The art & science of negotiation. Cambridge, MA: The Belknap Press of Harvard University Press, 1982.

15. Ulvila, J.W. Decisions with multiple objectives in integrative bargaining. (Doctoral dissertation, Harvard University, Graduate School of Business Administration, 1972). Dissertation Abstracts International, 1979, 40, 1594A. University Microfilms No. 79-20985.

16. Levy, Steven. Hackers: Heroes of the computer revolution. Garden City, NY: Anchor Press/Doubleday, 1984.

17. Brown, R.V. Personalized decision analysis as an expert elicitation tool: An instructive experience in information security policy (Report to OTA - Task 2) (Technical Report No. 85-9). Falls Church, VA: Decision Science Consortium, Inc., February 1985.

18. Brown, R.V., & Lindley, D.V. Plural analysis: Multiple approaches to quantitative research. Theory and Decision, 20, 1986, 133-154.

19. Brown, R.V., & Feuerwerger, P.H. A macromodel of nuclear safeguard effectiveness (Interim Report PR 78-6-80). McLean, VA: Decisions and Designs, Inc., March 1978.

20. Raiffa, H. Decision analysis. Reading, MA: Addison-Wesley, 1968.

21. Brown, R.V., Kahr, A.S., & Peterson, C.R. Decision analysis for the manager. New York: Holt, Rinehart, and Winston, 1974.

22. Hoffman, L.J., A research agenda for computer security risk analysis (draft). Report on a DOD-sponsored workshop, Washington, DC: The George Washington University, Department of Electrical Engineering and Computer Science, January 1986.

23. Brown, R.V. Assessment uncertainty and the firmness of information: A decision-oriented methodology. Falls Church, VA: Decision Science Consortium, Inc., May 1986.

24. Watson, S.R. An index of hazard from radioactive waste (Technical Report). McLean, VA: Decision and Designs, Inc., 1977.

25. Luce, R.D. Individual choice behavior: A theoretical analysis. New York: Wiley, 1959.

26. Hill, G.A. Societal consequences of malevolent situations: Implications for safeguards policy (Technical Report 81-8). Falls Church, VA: Decision Science Consortium, Inc., May 1982.

ACKNOWLEDGEMENT

The author wishes to express appreciation to Lance Hoffman, who provided stimulus to work on this topic and suggested the idea for the principal figure here; and to Granger Morgan for some basic insights into the problem.

"ADVICE MOST NEEDED..." THE ASSESSMENT AND ADVICE EFFORT

Deborah M. Claxton

Department of Defense
9800 Savage Road
Fort George G. Meade, MD 20755

Abstract

The intent of this paper is to briefly inform the reader of the controversy brought about by National Security Decision Directive 145. Also, it will present to the reader an informative report of the Assessment and Advice (A&A) effort being carried out by the Applications Systems Evaluations Office of the National Computer Security Center (NCSC). The opinion of the author is that the A&A effort is one of the best ways for the NCSC to address both the directive and the controversy. The paper will:

1) give a brief account of NSDD 145 and the Center.
2) describe the actual process of an A&A -
   - for federal agencies which may wish to have an A&A performed.
   - for training computer security evaluators who are assigned to an A&A team.
3) encourage management to continue the A&A effort -
   - for the benefits to the NCSC.
   - for the benefits to the federal government.

Introduction

There is nothing more difficult to plan, more doubtful of success, nor more dangerous to manage than the creation of a new system. For the initiator has the enmity of all who would profit by the preservation of the old system and merely lukewarm defenders in those who would gain by the new one. [1]

Niccolo Machiavelli is credited with making this enlightened observation in the early 1500's, but his accurate account of resistance to change is still very evident today. A Presidential Directive concerning the issue of information security in the federal government has stirred up resistance on many fronts.

NSDD 145

"With the federal government's need for computer security so acute and so obvious, it's a shame that the White House's effort to address the issue has become so mired in controversy."[2] This quote from an editorial in the weekly Government Computer News (GCN) expresses the contention brought about by National Security Directive (NSDD) 145 which was issued by the National Security Council on September 17, 1984, and signed by President Ronald Reagan. The directive, which is titled the "National Policy on Telecommunications and Automated Information Systems Security," states these policy objectives: 1) to assure the security of telecommunication and automated information systems that process and communicate classified and other sensitive national security information, and 2) to offer assistance in the protection of certain private sector information. The National Security Agency has been named as the leading authority for accomplishing these objectives.[3]

Opposition to this directive has many concerns, ranging from the American Civil Liberties Union's concern for freedom of information, through a government agency's security specialist who says, "We don't want someone else telling us what to do,"[4] to some in Congress who are upset that the policy was made through a directive designed by the National Security Council and signed by the President without public input rather than by legislation which would receive public hearings and a full debate in Congress. Mostly, criticism stems from the leadership position for information security given to the National Security Agency.[5] The GCN editorial concludes:

> Given the need for federal computer security, we hope the agency is up to the task. Failure could set back the whole process by several years and many federal agencies are already years behind in security measures.[6]

Actually, many government standards concerning computer security were available prior to NSDD 145. However, these policies are often ambiguous, outdated, or cite

---

[1] Niccolo Machiavelli, The Prince.

[2] "Security," Government Computer News, Editorial, 27 September 1985, p. 14.

[3] U.S., National Security Council, "National Policy on Telecommunications and Automated Information Systems Security," National Security Decision Directive 145 (17 September 1984).

[4] Eric Fredell, "Agencies Balk at Control Given NSA," Government Computer News, 27 September 1985, p. 19.

[5] Eric Fredell, "Security Directive Lambasted," Government Computer News, 19 July 1985, p. 1.

[6] "Security," GCN editorial.

conflicting information and their ineffectiveness is evident by the lack of security in the computer systems of the federal government.

The Office of Management and Budget's (OMB) Circular A-71 requires that computer systems with sensitive applications be certified and accredited. The National Bureau of Standards' (NBS) Federal Information Processing Standards Publication (FIPS PUB) 102 details procedures for certification and accreditation for federal agencies and lists more than eighty federal computer security policies and guidelines. Numerous Department of Defense (DoD) regulations also exist outlining security requirements, safeguards for classified information, and modes of operation. Susan Menke reports the ineffectiveness of these requirements in an article in Federal Times:

> In the past, OMB and GAO have tried with mixed success to force everyone to think hard about the risks and consequences (of computer and computer information loss)....Many agencies remain overwhelmed by conflicting, overlapping security directives and so far haven't made much progress....Others hold a cynical laissez-faire view...that they'll roll with the punches when something valuable gets stolen.[7]

NSDD 145 points out that the nation's security is in jeopardy if the telecommunication and automated information systems which process national security-related information continue to operate as they have in the past. "The technology to exploit these electronic systems is widespread and is used extensively by foreign nations and can be employed as well by terrorist groups and criminal elements."[8] With all the policy and regulations concerning information security that have been available, no one agency has had the responsibility to foster computer security. NSDD 145 has directed that NSA be responsible for aiding agencies that process national security information, and now that NSA has been given this responsibility, the controversy spreads.

To make NSDD 145 work and ease the apprehensions of the great opposition, the National Security Agency has plenty to do. The ability to influence others toward greater information security must be used with aboveboard procedures. The biggest of the concerns, that in some cases borders on paranoia, is of NSA's being the Orwellian "Big Brother." The GCN editorial, "Security," further explains the "fear of Big Brother" and suggests how the fear might be overcome:

> Probably most of those with an interest in better security measures would have been happy if authority came from anyone - anyone but NSA, that is....If NSA is going to smooth the waters, gain the trust of the agency ADPers whose cooperation is vital to this program's success and hold off revision minded congressmen, it will have to...cooperate with, not dictate to, the agencies, it will have to work closely with Congress and the private sector... it will have to operate much more openly than it has ever done before....[9]

NSA must reemphasize this point: it is the responsibility of each agency, whether defense or civil, to determine where and what its most valuable assets are, and what the consequences of exposure of those assets would mean to each agency.The point was clarified by Assistant Secretary of Defense, Donald C. Latham, when he testified before a House subcommittee that the directive:

> Does not make NSA the government's oversighter of all civil agencies...and allow them into everybody's computers and tell them what to do....only where appropriate will there be any assistance to the civil sector. (The assistance from NSA will be advice and information in most cases)....Implementation of security measures is the responsibility of the federal departments and agencies, not the director of NSA or the DoD.[10]

Sensitive applications must be certified and accredited; moreover, the process of accreditation is an integral part of system security. NSA's role is to provide guidance to departments and agencies.

The technical experts for information security are available at NSA, and their knowledge is available for those who need help with computer security issues in their own agencies. Once these agencies, the customers, have requested computer security assistance from NSA, the technical experts must keep in mind that their job is to provide a service to those customers. Open lines of communication and a professional attitude on the part of the NSA experts will add much to the effort to allay the opposition's fears. A large part of the

---

[7]Susan M. Menke, "Security is More a Human Issue Than a Technical One," Federal Times, 4 November 1985, p. 18.

[8]NSDD 145.

[9]"Security," GCN editorial.

[10]Eric Fredell, "Latham: NSDD 145 Does Not Restrict Agency Roles," Government Computer News, 11 October 1985, p. 16.

success with which NSA fulfills its mission as the leader for fostering information security is dependent upon the National Computer Security Center (NCSC), the organization within NSA where the computer security experts work.

The Center

With NSDD 145, the Department of Defense Computer Security Center (DoDCSC) became the National Computer Security Center; however, more than just the name has been changed. The Center's mission and responsibility have expanded to include not only the DoD but also the civil sector (non-DoD departments and agencies of the Executive Branch) of the Federal Government where appropriate. ("Where appropriate" means having systems which deal with classified information or other national security-related information.)

The Department of Defense Computer Security Center (DoDCSC) was formed in January, 1981, with a major goal stated in its charter of "encouraging widespread availability of trusted computer systems by those who process classified or other sensitive information." The Center has followed a strategy to improve the level of data security in computer systems throughout the Department of Defense by various efforts. The strategy has been to emphasize the need to install state-of-the-art secure "trusted" systems and to promote the availability of those systems.[11]

The task of the new mission is a tremendous one, but the effort to contact the more than two million federal employees who need to become aware of computer security has begun. Two upper level management representatives from the Center have been circulating to various federal agencies in an attempt to open the lines of communication with federal managers. The purpose of this contact has been to provide information about the Center and to identify computer systems used by the organization. A new "desk officer" program has also begun with the purpose of providing a point of contact for the federal agencies in their dealings with the Center.

It is obvious, however, that the lack of both resources and time will hinder the Center's ability to reach two million federal employees. The enormity of the task has been described as such by the Center:

Although NSDD 145 gives NSA the responsibility for automated information systems processing national security related information, we at the NCSC will only be able to help those civil agencies that process national security related information <u>and</u> request our assistance.[12]

An Office Level Management Review (OLMR) report from the Applications Systems Evaluations Office advocates better use of existing resources to address the NSDD 145 tasks. The report states that a greater service will be provided by giving sound advice and support to many projects rather than devoting resources to long term, in-depth analysis of a few systems. The best means of providing support to many is through short term undertakings. Short term efforts will benefit not only the customer, but also the Center.[13] Benefits to the customer would be giving the customer a service that is much needed, and helping the customer agency build its own computer security expertise. Benefits to the Center would be building the Center's own knowledge base, on-the-job training for new computer security analysts, and improving public relations for the Center.

Short term efforts which are available for both DoD and civil sector customers include:

1) introductory briefings: information on the threats and vulnerabilities of untrusted computer systems and how to reduce risks, presentation of services which can be provided, and educational information;

2) technical consultations: meetings arranged at the request of the customer to discuss particular areas of concern or general computer security issues; and

3) Assessment and Advice (A&A) studies: on-site technical analyses in support of any phase of a project.[14]

---

[11]U.S., Department of Defense, DoD Computer Security Center, <u>Department of Defense Trusted Computer System Evaluation Criteria</u>, CSC-STD-001-83 (15 August 1983), p. 1.

[12]Letter to Ms. Jean Smith of the Congress of the United States, Office of Technology Assessment from the National Computer Security Center, (3 December 1985).

[13]"Office Level Management Review (OLMR) Summary Report," 10 December 1985, National Computer Security Center.

[14]Briefing at the National Computer Security Center, Ft. Meade, MD, 22 November 1985.

Again, these short term endeavors must be accelerated to create the most benefit to everyone concerned. In particular, concentration upon the Assessment and Advice effort can help generate the greatest service in the least amount of time to customers, and can provide the Center's analysts with the knowledge of systems currently being used throughout government agencies.

A&A's

An Assessment and Advice effort is not an inspection, certification, or risk analysis but rather is a technical analysis of the computer security posture of a particular system and advice to the customer about vulnerabilities of the system. The A&A will identify security problems and propose reasonable solutions that are achievable by the customer. In addition to immediate suggestions to improve computer security, long term planning suggestions are provided.[15] These suggestions will enable the customer to assume a self-help posture and to do more for themselves with the Center providing counseling and tools to help them.[16]

Preliminary Planning

The process of an actual A&A is a structured operation. Preliminary planning begins with tasking from the customer in writing. This is important for a clear understanding of what is expected and what will be done by all involved. All "buzzwords" must be clearly defined, especially the fact that an assessment is not a full-fledged certification. The difference between these two technical analyses of systems, one analyst explained, is that during an assessment, the computer security evaluators consider the system documentation, procedures and personnel as witness to the security of the system; whereas, in a certification, the security of the system must be proven, tested and validated by the analysts.[17] Also defined in the tasking should be the policy or criteria with which the system will be compared. Good communication from the beginning of the A&A effort is very important.

Acceptance of the tasking should clarify in writing all that will be done for the customer with an outline of the milestones for completing the process of the A&A. The customer must provide information to the team of computer security analysts who will perform the A&A. The information needed consists of documentation and manuals pertaining to the system to be assessed and a questionnaire or survey form provided to the customer, completed promptly by customer personnel knowledgeable of the system and returned to the team of analysts at the Center. A generic questionnaire is currently being produced by analysts in the NCSC for use in A&A's. Scheduling for the physical site visit should take into consideration the preparation and review of documentation which the team must make. Classification or some type of protection for the final assessment report is necessary for the confidentiality of the information between the customer and the Center. Some customers may need special charters to protect the information from Freedom of Information Act inquiries.[18] The customer should be advised to submit this information to the team so that proper classification procedures can be followed.

Having accepted the task, the team of analysts must keep an open line of communication flowing between the customer agency and the Center. It is very important that the customer furnish necessary information mentioned as quickly as possible. Without the documentation, manuals and the completed survey, the evaluators cannot begin to familiarize themselves with the system to be assessed. Friendly communication will encourage the customer to deliver the materials promptly. Once the materials have been received, a quick call or note keeps the customer informed and lets them know that the A&A is progressing. If scheduled milestones or times must be adjusted, being honest with the customer and not making promises which cannot be kept are part of the professional attitude which the team from the Center must present. The team's own management must be informed of the progress of the A&A, with what the team is doing and the time frame for activities planned.

The actual job at hand for the assessment team is to become familiar with the system before the on-site visit. This preparation allows team members to ask intelligent, well-thought out questions of site personnel and prevents the necessity of being briefed "from scratch" by them. If the system is a large one, tasks might be broken down, and smaller teams formed to concentrate on specific technical areas and questions concerning these areas. A point which the analysts must remember during the review of material is that they must not make premature judgments of the system which are unsupported by facts. Unclear areas in documentation may easily be explained by the site personnel if the analysts have not already formed an adverse opinion. In addition to computer security knowledge, each member of the A&A team must also be

---

[15]Ibid.

[16]OLMR Summary Report.

[17]Interview at the National Computer Security Center, Ft. Meade, MD, 4 September 1985.

[18]U.S., Department of Commerce, National Bureau of Standards, Guideline for Computer Security Certification and Accreditation, Federal Information Processing Standards Publication 102 (1983), p. 64.

armed with skills for briefings, interviewing, and report writing. Advanced preparation by all team members benefits the Center by presenting a professional appearance to the customer. Preparation thus increases the customer's confidence in the conclusions and recommendations made by the Center's computer security specialists during the physical site visit and in the final assessment report.

On-site Visit

The on-site visit itself is of short duration, from two to five days. Here, the analysts confer with customer personnel who know the system. The physical site visit gives the technical team the opportunity to examine the environment in which the system actually operates and the procedures which personnel follow when using the system. The existence of most physical and administrative controls such as locks, guards and written logbooks can be seen during the visit. Controls internal to the machine such as passwords and audit trails can be shown in demonstrations and verified during the interviews with customer personnel. The intent is simply to show the existence or lack of proper controls. Active penetration testing as performed in a certification effort is beyond the scope of an A&A. The visit consists of several meetings for which the customer must arrange facilities and personnel.

Generally, the visit commences with an in-briefing. The in-brief consists of a reconfirmation of the tasking, what is to be done during the visit, and general information on the final report. Following this briefing, a session is led by the customer who presents a general overview of the system, general concerns to be covered, a visit to the computer and terminal areas, and demonstrations of the system.

The team then proceeds to interview a variety of site personnel, whether they be users, operators, programmers, managers, or system security officers, who are knowledgeable of technical aspects of the system. Interviews are a form of interpersonal communication and subject to the usual problems of misunderstandings. Team members must strive to prepare well for the interview so that they already know the answers to the questions that they ask and are, therefore, simply verifying information that they have already gathered. The objective of the reviewer during the interview is to solidify an informed opinion which is then presented in the final assessment report. Again, advanced preparation aids the team in obtaining proper information and is crucial to a successful assessment.[20]

[20]Ibid., p. E-1.

Once all interviews have taken place, the computer security team meets in a private session to discuss their findings and concerns. Since a team decision is made concerning the security posture of the system which they have just examined, the team members work as a body to formulate the decision and suggestions which are incorporated into an out-brief during the visit, and finally, in the written assessment report.

The out-briefing is essentially a short form of the final report. The findings, conclusions and recommendations determined by the technical team are addressed. Now, the customer knows what to expect in the final report; there will be no surprises. Conclusion of the on-site visit leaves only paperwork to be done by the team.

Final Report

The final assessment report containing information gathered during the preparation period and the physical site visit is written and delivered to the customer as quickly as possible. Time frames agreed upon in initial acceptance of tasking normally target completion of the report within 30 - 60 days after the site visit. The final report presents an informed opinion of the security posture of the system and computer security advice to the customer. A basic part of the report contains information on how the system meets the policy requirements to which it should conform. Whether the requirements be a Criteria class or the customer's own standards, the policy to consider has been agreed to by management in the initial tasking.

Contents of the report consist of how the current system "stacks up" to the policy, concerns specific to the system, possible solutions to any vulnerabilities found, guidance for the future of the system, and encouragement for the customer to continue the pursuit of computer security. Each of these points should be covered during the out-brief for the customer, and the final report just expands on those points. Classification guidance for the report which was specified by the customer agency must be followed. Again, credibility for the Center grows with a timely completion of the report and its submission to the customer.

Conclusion

The Assessment and Advice effort is an excellent tool which the Applications Evaluation Office can employ in fulfilling the mission of NSA and the Center defined in NSDD 145 and attempting to quell the fears of those who oppose that directive. Continuation of these short term projects will quickly aid in the Center's goal of being viewed as a "help not a hindrance."[20] In addition, the knowledge base of the Center itself will also benefit.

"Organizations are always involved in some process of transformation....Healthy organizations are responsive to feedback, using it in part, as a basis for future messages, policies and actions."[21] Hopefully, A&A customers will be open to suggestions that the Center makes; and hopefully, from those customers that are served, word will spread that the Center is doing a service and doing it well. Hopefully, the old adage, "Advice most needed, seldom heeded," will not be the standard which the agencies follow. A willingness to cooperate and work together helps everyone get on the right track toward information security.

Summary

"All organizations must remain relatively open in order to survive. Organizations cannot exist as static systems...."[22] This fact is evident even within an organization as large as the federal government of the United States. As the National Security Agency and the National Computer Security Center address the critical need for computer security in the information systems of federal agencies, it is hoped that those agencies will have the ability to accept the advice which they are given and to facilitate needed changes.

> Real security for electronic information does not result from rules on pieces of paper, assignments of people, hardware facilities or software systems.... Security depends on people's being willing to comply...to the means selected for protection.[23]

NSA and the Center can ease the resistance to change by employing aboveboard procedures and by quickly giving help where needed. Short term efforts, in particular the Assessment and Advice program, are the best means for accomplishing this goal. Computer security in the information systems of the federal government not only can happen, it must happen, for the security of the nation is dependent upon it.

---

[20]OLMR Summary Report.

[21]Patricia Hayes Bradley and John E. Baird, Jr., Communication for Business and the Professions, 2nd ed. (Dubuque: William C. Brown Co. Publishers, 1983), p. 18 - 21.

[22]Ibid, p. 20.

[23]James A. Schweitzer, Managing Information Security: A Program for the Electronic Information Age, (Boston: Butterworth, (Publishers) Inc., 1982), p. 2.

Bibliography

Bradley, Patricia Hayes, and Baird, Jr., John E. Communication for Business and the Professions. 2nd ed. Dubuque: William C. Brown Co. Publishers, 1983.

Campbell, Robert P. "Tech Faults, Small Demand Make Security Buys Risky." Government Computer News, 27 September 1985, p. 19.

"Center Works to Increase Safeguards." Government Computer News, 27 September 1985, p. 24.

Couch, Walter R. "Agencies Need to Identify Sensitive Applications." Government Computer News, 27 September 1985, p. 40.

Fredell, Eric. "Agencies Balk at Control Given NSA." Government Computer News, 27 September 1985, p. 19.

Fredell, Eric. "DoD's Latham Defends New NSA Security Role." Government Computer News, 11 October 1985, p. 1.

Fredell, Eric. "Latham: NSDD 145 Does Not Restrict Agency Roles." Government Computer News, 11 October 1985, p. 16.

Fredell, Eric. "Security Directive Lambasted." Government Computer News, 19 July 1985, p. 1.

Fisher, Dalmar. Communication in Organizations. St. Paul: West Publishing Co., 1981.

Fisher, Royal P. Information Systems Security. Englewood Cliffs, NJ: Prentice - Hall, Inc., 1984.

Greene, Joseph S., Jr. "DoD Overview: Computer Security Program Direction." Proceedings of the 8th National Computer Security Conference. Gaithersburg, MD: n.p., 1985, pp. 6-10.

Harvey, L. James. "Flexibility to Face Change Computer News, 8 November 1985, p. 31.

Hoffman, Lance J. "PC Software for Risk Analysis Prove Effective." Government Computer News, 27 September, p. 58.

"ICST Specialist: Security Depends on Environment." Government Computer News, 27 September 1985, p. 28.

Levine, Arnold S. "Conference Looks at Results of Security Efforts." Government Computer News, 8 November 1985, p. 64.

Levine, Arnold S. "Energy Explains Dept. Security Problems, Solutions." Government Computer News, 8 November 1985, p. 65.

Martin, James. Security, Accuracy, and Privacy in Computer Systems. Englewood Cliffs, NJ: Prentice Hall, Inc., 1973.

Menke, Susan M. "Security is More a Human Issue Than a Technical One." Federal Times, 4 November 1985, p. 18.

McLoughlin, Glenn. "Congress Addresses Crime and Security." Government Computer News, 27 September 1985, p. 19.

National Computer Security Center. Letter to Ms. Jean Smith of the Congress of the United States, Office of Technology Assessment. (3 December 1985).

National Computer Security Center. "Office Level Management Review (OLMR) Summary Report." (10 December 1985).

Parker, Donn B. Computer Security Management. Reston, VA: Reston Publishing Co., Inc., 1981.

Parker, Donn B. Fighting Computer Crime. New York: Charles Scribner's Sons, 1983.

Schweitzer, James A. Managing Information Security: A Program for the Electronic Information Age. Boston: Butterworth (Publishers) Inc., 1982.

"Security." Government Computer News, editorial, 27 September 1985, p. 14.

Stahl, Taro and Shumar, Chuck. "Restricting Access is New Challenge to Mgmt." Government Computer News, 27 September, 1985, p [illegible]

U.S. Department of Commerce. National Bureau of Standards. Guideline for Computer Security Certification and Accreditation. Federal Information Processing Standards Publication 102 (1983).

U.S. Department of Defense. DoD Computer Security Center. Department of Defense Trusted Computer System Evaluation Criteria, CSC-STD-001-83 (15 August 1983).

U.S. National Security Council. "National Policy on Telecommunications and Automated Information Systems Security," National Security Decision Directive 145 (17 September 1984).

Wong, Kenneth K. Risk Analysis and Control. Rochelle Park, NJ: Hayden Book Co., Inc., 1977.

A MODEL OF INFORMATION

David Sutherland

Odyssey Research Associates, Inc.

1283 Trumansburg Road
Ithaca, New York 14850

The highest level requirements on a secure computing system are usually stated in terms of information, that is, they state that certain information must not be obtained by certain individuals on the system. Formal models of computer security to date have concerned themselves largely with restrictions on the movement of data. While these restrictions capture part of the high level requirements of secure systems, they do not capture all of them, as witnessed by the fact that there is a distinction made between "formal modeling" and "covert channel analysis". True formal verification of a secure system would incorporate the security violations usually covered by "covert channel analysis" into the formal model of the system.

In this paper we present a simple model of information and inference, give a generic instantiation of the model to state machines, apply the instantiation to a simple example, and discuss the relationship between the state machine instantiation and the Goguen-Meseguer non-interference model.

What is information? What does it mean to infer information from other information? We will start with a few naive answers to these questions.

1 Information

In answer to the first question, someone who was used to thinking in terms of formal specifications of systems might answer "the values of some collection of state variables". What's wrong with this answer? One problem with this answer is that it's not general enough. Most instances of information channels involve observing the values of some collection of state variables over the course of time; no one instantaneous value contains the information transmitted. Suppose we amended the above answer to "information is the history of some collection of state variables" where "history" means "history from time 0 to some time t". One problem with this second answer is that, in order to apply it to a system, the system must be expressed as a state machine. Not only does this force us to use a certain representation, but our analysis of information in the system might be unduly sensitive to, say, what state variables we choose. What we wish to do at this point is generalize the second answer so that it's independent of how the system is represented. To do this, we'll look at the second answer in a little more detail.

For the purpose of this discussion we will say an abstract state machine consists of:

1. A set of states
2. A set of possible initial states
3. A set of _state transformations_, by which we mean a function from states to states.

The set of states is usually defined by giving a set of _state variables_, each of which has a certain type; a "state" is then an assignment of each state variable to a value in its type.

How do we imagine such an abstract machine actually "running"? We imagine the machine starting out in one of the possible initial states, and then changing state over the course of time as various state transformations are applied. For each possible initial state and each sequence of transformations applied, we get a sequence of states that the machine passes through. Thus, an abstract machine defines a set of possible sequences of states that the machine can pass through. We will call these sequences _possible execution sequences_. We will regard time as being measured in terms of the number of state changes the machine has passed through, so "state at time 0" refers to the initial state of the machine, "state at time 1" refers to the state of the machine after one application of a state transformation, and so on.

Given a collection of state variables V and a time T, what do we mean by "the history of the state variables in V from time 0 to time T"? This "history" is actually a function (call it h) whose domain is the set of possible execution sequences. Given a possible execution sequence S, h(S) is a finite sequence of length T + 1 such that:

1. The ith entry of h(S) is an assignment of each state variable in V to a value in its type.
2. For each state variable v in V, the value assigned to v by the ith entry of h(S) is the same value assigned to v by the ith entry of S.

In other words, H takes the first T + 1 entries of S and extracts out the assignments of the variables in V.

We generalize the above scheme in the following way: we represent a system as a set of _possible worlds_; this corresponds to the set of possible execution sequences for an abstract state machine. A particular "piece of information" about the system is represented as a function whose domain is the set of possible worlds; we will call such functions <u>information functions</u>. An information function can be thought of as a certain "view" of the system; in a given possible world w, an information function returns what is "seen" of w by someone "looking at" the information function.

2 Inference

We now return to the second question posed at the beginning of the section: what does it mean to infer information from other information? In terms of the formalism developed above, what does it mean to infer something about the value of one information function from the value of another information function? To answer this question, we will imagine a user who:

1. knows what the set of possible worlds W is (this corresponds essentially to knowing how the system is designed);

2. knows what information function he is seeing (call it f1) (this corresponds to the user knowing what his interface to the system is. If, for example, a user could see a terminal but had no idea how the output appearing on the terminal was being generated, he would be able to deduce very little about the system.);

3. knows what information function he wishes to deduce something about (call it f2);

4. is "in" some possible world w, and knows the value of f1(w) (call it x).

What can the user infer about f2(w)? Since he knows that f1(w) = x, he can deduce that w is in the set of all possible worlds y such that f1(y) = x. Call this set S. On the basis of the above knowledge, all the user can deduce about w is that it is in S. From this, he can deduce that f2(w) is in the image of S under f2. Call this image T. If there is some value z in the range of f2 that is achieved in some possible world but which is not in T, then the user has actually gained some information about f2(w), namely, he at least knows that it is not equal to z. If, on the other hand, every value z in the range of f2 that is achieved in some possible world is in T, then the user knows nothing more about f2 than he could have inferred on the basis of knowing W and f2 alone. This leads us to the following definition:

> Given a set of possible worlds W and two functions f1 and f2 with domain W, we say that <u>information flows from f1 to f2</u> if and only if there exists some possible world w and some element z in the range of f2 such that z is achieved by f2 in some possible world but in every possible world w' such that f1(w') = f1(w), f2(w') is not equal to z.

Having given this somewhat complicated but reasonably motivated definition, the first thing we will do is note that it is equivalent to a much simpler statement.

<u>Proposition</u>: Given W, f1 and f2 as above, information does <u>not</u> flow from f1 to f2 if and only if the function f1 x f2 from W to the cross product of the images of f1 and f2 is onto.

<u>Proof</u>: Suppose information does not flow from f1 to f2; we wish to show that f1 x f2 is onto image(f1) x image(f2). Let (x,y) be an element of the cross product. Since x is in the image of f1, there exists a possible world w1 such that x = f1(w1). Likewise, there exists a possible world w2 such that y = f2(w2). If we take the negation of the above definition with w = w1 and z = y, we get that there must exist a possible world w' such that f1(w') = f1(w1) = x and f2(w') = y. In other words, (f1 x f2)(w') = (x,y). Since (x,y) was arbitrary, f1 x f2 is onto.

Conversely, suppose f1 x f2 is onto. Let w be a possible world and z an element of the range of f2 that is acheived by f2 in some possible world (in other words, z is an element of image(f2)). Then (f1(w),z) is an element of image(f1) x image(f2) and so there exists a possible world w' such that (f1 x f2)(w') = (f1(w),z), i.e. f1(w') = f1(w) and f2(w') = z.

<u>Corollary</u>: the information flow relation is symmetric.

The corollary seems somewhat surprising at first glance, since information flow is not usually thought of as being necessarily a two-way street. However, consider the following scenario: a system is designed so that every character which is typed at keyboard K is echoed to screen S. Let f1 be the information function which, given a possible world, returns the sequence of all characters typed on K in that world, and let f2 be the information function which, given a possible world, returns the sequence of characters displayed on S in that world. Information is obviously being transferred from K to S, and so we would expect to find that, according to the definition above, information flows from f1 to f2. This is exactly what we do find. By the corollary, we will also find that information flows from f2 back to f1, i.e., it will find that, knowing the value of f1, one can infer something about f2. But this is in fact true! If one knows the design of the system, and one knows what has been typed at K, one knows something about what shows up on S. A problem arises, however, if we try to assign security levels to information functions and require that information always flow "up." If we assign K a level "lo" (say, because only "lo" individuals are allowed to type at it) and S a level "hi" (say, because only "hi" people are allowed to view it), we will find a flow in both directions, violating the security requirement, despite the fact that the system as described seems secure. The problem is not in the definition of information flow but rather in the choice and labeling of information functions. S may be a "hi" terminal, but the information it gets from K is not "hi" information, and thus should not be labeled as "hi." Actually, the only information which should be labeled as "hi" is information which comes from high sources. We will discuss this further below when we instantiate the information model to state machines.

3 Security Conditions

We now have a model of information and a definition of information flow. What we need to complete the model is a definition of "secure". Informally, a system is secure if nobody can get information he's not entitled to. How can we express this in the above formalism?

First of all, each "piece of information" which has restrictions on who may "get" it is represented by an information function. Second, each "entity" on the system which has restrictions on what information it can "get" is represented by an information function corresponding to the entity's "view" of the system. We will denote the set of information functions corresponding to pieces of information and entities by IF. We represent the restrictions on which entities are entitled to "get" which pieces of information by a binary relation "legal_to_get" on IF.

How can we formally express the notion of an entity E "getting" a piece of information I? If f1 is the information function corresponding to E and f2 is the information function corresponding to I, we interpret "E 'gets' I" to mean "information flows from f2 to f1".

Under these interpretations, the informal statement of security given at the beginning of the section is formalized as

> For all f1 and f2 in IF, if information flows from f2 to f1 then legal_to_get(f1,f2)

Readers familiar with formal computer security may at this point be asking "Where are the security levels in this model?" The answer is that security levels are a particular instance of the model. We could assign security levels to the functions in IF and define legal_to_get(f1,f2) if and only if the level of f1 is less than or equal to the level of f2. This is just one possible way of defining legal_to_get; the model allows for others, e.g. discretionary access restrictions.

4 Summary of the Model

In this section we summarize the information model briefly. An instance of the model consists of:

1. A set W of possible worlds
2. A set IF of functions with domain W
3. A binary relation on IF, legal_to_get

Such an instance is secure if and only if for every f1 and f2 in IF, if information flows from f2 to f1 then legal_to_get(f1,f2) (where information flow between functions with domain W is defined as above).

5 State Machine Instantiation

In this section, we instantiate the information model given above to a state machine. We begin by defining what we will mean by a state machine.

5.1 State Machines

"State machine" will mean a non-deterministic finite automaton with null moves except that the state space of the automaton is not required to be finite. In other words, a state machine consists of a set of states, a non-empty set of initial states, an alphabet, and a set of "arrows." Each "arrow" starts at one state and points to another; an arrow may be labeled with a single element of the alphabet or it may be unlabeled. The "operation" of the machine is to start at an initial state and change state in steps, with each state change accompanied by one or no letters of the alphabet.

We now add a few extra structures and some additional axioms. First of all, at this point we will stop using the word "alphabet" and refer to the set we formerly called the alphabet as the signals of the state machine. The signals of the machine are partitioned into the input signals and the output signals. There is also a set of security levels, partially ordered by a relation <=, and a function from signals to levels.

We require that for every state of a state machine and every input signal, there is an arrow which starts at the given state and is labeled with the given signal. In other words, it is always possible for a state machine to receive a given signal (even if its only response is to remain in its former state). We also require that for every state of the machine there is an arrow which starts at the given state which is not labeled with an input signal. In other words, it is always possible for the state machine to "go ahead", even in the absence of an input.

5.2 Instantiating the Information Model

We wish to interpret the information model in terms of state machines. In other words, we want to give a procedure which takes a state machine and returns the corresponding information model. Security for the state machine is then defined simply as security for the corresponding information model. Thus, we need to give a procedure which, given a state machine, gives a corresponding set of possible worlds, a set of information functions for that set of worlds, and a binary relation on those information functions defining what information flows between them are legal. We now describe this procedure.

Suppose we have a state machine described by:

1. A set of states S
2. A set of initial states I
3. A set of input signals X
4. A set of output signals Y
5. A set of "arrows" A (we won't give a completely formal definition of what an "arrow" is, as it would be more obfuscatory than anything else).
6. A set of levels L with partial ordering <=
7. A function from X U Y to L

The set of possible worlds we associate with this machine is the set of finite sequences H = { H[i] | 0 <= i <= length(H) - 1 } such that:

1. Each H[i] is either a state or a signal
2. No two consecutive entries in the sequence are both signals
3. H[0] is an initial state
4. If H[i] and H[i+1] are states, there is an unlabeled arrow from H[i] to H[i+1]
5. If H[i] is a state and H[i+1] is a signal, there is an arrow starting at H[i] labeled with H[i+1]
6. If H[i] is a state, H[i+1] is a signal and H[i+2] is a state, there is an arrow from H[i] to H[i+2] labeled with H[i+1]

There is actually an important reason why we have chosen finite sequences rather than infinite sequences as our possible worlds. Under the above instantiation, a possible world is literally a possible run of the system up to a given point in time. Thus, any inference made in such a world can only take into account the information about the world which has manifested itself up to the point in time being considered. We can think of such finite worlds as being initial segments of some real, complete possible world (i.e. an infinite sequence), but any inference must be made at some finite point in it. This choice literally affects whether some systems are formally secure or not, and the choice we have made seems to make the "right" systems formally secure.

For each level l in L, we define two information functions over the possible worlds defined above:

1. view(l) is the function which, given a possible world H as above, returns the subsequence of H consisting of those signals s whose levels are <= l
2. hidden_from(l) is the function which, given a possible world H, returns the subsequence of H consisting of those input signals s whose levels are not <= l

Finally, we specify that it is illegal for information to flow from hidden_from(l) to view(l) for any l.

This choice of information functions and illegal flows is based on the following picture: there is a collection of "entities" external to the machine which interact with it, and each of these entities has a level. It is assumed that each entity only "knows" some signals going into and coming out of the machine, and that an entity of level l is allowed (by procedural safeguards or whatever) to see at most all of the signals that go into or out of the machine whose levels are <= l. The choice of illegal flows simply reflects the policy that an entity of level l should not be able to deduce from what it is legal that he see anything about what it is not legal that he see. Notice that, according to the instantiation, there is nothing wrong per se in a low level entity being able to deduce a high level output signal. If a high level output signal is unconnected to any high level input signals, then it is not a violation of security for a low level entity to see it. If, on the other hand, such a high level output has some connection to a high level input, then this connection will presumably be reflected as an information flow and thus an information flow will be found from the high inputs to the low entity, violating the policy as stated.

6 An Example

In this section we give a simple example of the use of the state machine instantiation of the information model. We will first describe the machine informally.

The machine is a simple message-passing system. It consists of a collection of "ports" and a queue of "message entries."

Each port is labeled with a security level indicating what level entities external to the machine can access the port. Ports can input messages to the machine, which get put on the queue in a message entry. The message entry also contains the information of which port the message came from. The intended destination of the message is contained in the message. When a message entry comes to the head of the queue, one of two actions is taken: (1) if the level of the destination port is greater than or equal to the level of the source port, the message is output to the destination port and the message entry is removed from the queue; (2) otherwise, the message entry is removed from the queue and no output occurs.

We now describe the above machine formally. We denote the set of ports by P, the set of messages by M, the set of security levels by L, the partial ordering on L by <=, the function which takes a port and returns its level by lvl (a function from P to L), and the function which takes a message and returns its destination by dest (a function from M to P).

The state of the machine is a sequence of pairs (m,p) where m is in M and p is in P. The initial state of the machine is the empty sequence.

A signal of the machine is a triple (m,p,x) where m is in M, p is in P and x is in the set {input,output}. Such a signal is an input signal if x is "input" and an output signal if x is "output". If x is "input", p is the port from which the signal came. If x is "output", p is the port to which the signal goes. The level of a signal (m,p,x) is lvl(p).

The arrows of the machine are as follows:

- For each state s, each m in M and each p in P, let s' be the sequence s with the pair (m,p) prepended; there is an arrow from s to s' labeled with (m,p,input).

- For each state s, each m in M and each p in P, let s'' be the sequence s with the pair (m,p) appended:

  * If lvl(p) <= lvl(dest(m)), there is an arrow from s'' to s labeled with (m,dest(m),output).

  * If lvl(p) is not <= lvl(dest(m)), there is an unlabeled arrow from s'' to s.

- There is an unlabeled arrow from the empty sequence to itself.

We will now prove that this machine meets the security condition of the state machine instantiation of the information model.

Fix a level l in L. We wish to prove that no information flows from hidden_from(l) to view(l). First, we will define a function R from states to states as follows: if s is a state (i.e. a sequence of message-port pairs), R(s) is the subsequence of s consisting of the entries (m,p) such that lvl(p) <= l.

We want to examine what happens to a possible world of the machine (i.e. a finite sequence of states and signals meeting the conditions described in the previous section) when we apply R to every state in it. To do this, we must examine the effect of R on the initial state of the machine and the pairs of states at the ends of the various arrows.

The initial state of the machine is the empty sequence, and R of the empty sequence is the empty sequence. Thus, R of the initial state is the initial state.

Suppose there is an arrow from s to s' labeled with (m,p,input); s' must therefore be s with (m,p) prepended. What do R(s) and R(s') look like? If lvl(p) <= l, R(s') is R(s) with (m,p) prepended, and so there is an arrow from R(s) to R(s') labeled with (m,p,input). If lvl(p) is not <= l, R(s) equals R(s'). In other words, if the arrow corresponds to an input signal of level <= l, the states at the ends of the arrow are mapped to states at the ends of an arrow corresponding to the same input signal; in this case we will say that R "preserves" the arrow. If the arrow corresponds to an input signal whose level is not <= l, the states at the end of the arrow are mapped to the same state; in this case we will say that R "masks" the arrow.

Suppose there is an arrow from s'' to s labeled with (m,dest(m),output). s'' must therefore be s with (m,p) appended for some p such that lvl(p) <= lvl(dest(m)). In this case, if lvl(p) <= l then R(s'') is R(s) with (m,p) appended, and there is an arrow from R(s'') to R(s) labeled with (m,dest(m),output). Again, we say that R preserves the arrow. If lvl(p) is not <= l, R(s'') equals R(s), and we say that R masks the arrow.

Suppose there is an unlabeled arrow from s'' to s. There are two possibilities for s'' and s. First, s'' and s can both be the empty sequence, in which case R(s'') = R(s) = the empty sequence and we say that R preserves the arrow. Second, s'' can be s with (m,p) appended for some p such that lvl(p) is not <= lvl(dest(m)). In this case, if lvl(p) <= l then R(s'') is R(s) with (m,p) appended, and there is an unlabeled arrow from R(s'') to R(s). Again, we say R preserves the arrow. If lvl(p) is not <= l, R(s'') equals R(s) and we say that R masks the arrow.

What happens when we take a possible world H and apply R to every state in it? We can think of H informally as consisting of a sequence of arrows, with the first arrow starting at the initial state and with the ending and starting states of consecutive arrows matching. Each such arrow will either be preserved by R or masked by R. If we "throw away" the arrows that are masked by R, we get a new possible world of the machine. Call this world RH. What is the relationship between RH and H? RH is essentially H with all input signals whose levels are not <= l removed. In addition, all outputs and state changes resulting from such signals (i.e. being queued, being dequeued) are also removed. On the other hand, all of the input and output signals of level <=l are the same (the proof of this relies on the fact that the machine only allows signals from a given port to be output to ports of equal or greater level). In other words, view(l)(RH) = view(l)(H) while hidden_from(l)(RH)= the null sequence. Given any sequence s of input signals of level not <=l, we can add them on to the end of RH to get a possible world H' such that view(l)(H')= view(l)(RH) and hidden_from(l)(H')=S. Since H was arbitrary, this shows that for any value v1 in image(view(l)), any any value v2 in image(hidden_from(l)), there exists a possible world H' such that view(l)(H')= v1 and hidden_from(l)(H')= v2. In other words, view(l) x hidden_from(l) is onto image(view(l)) x image(hidden_from(l)) so there is no information flow between them. Since l was arbitrary, this proves the security condition.

7 Connection with the Goguen-Meseguer Model

We can think of the above proof as taking place in two stages. We start with an arbitrary possible world H, and we show that:

1. We can replace H by RH so that there are no signals of level not <= l, without changing any signal of level <= l.

2. We can replace RH by H' to make the signals of level not <= l anything we want, without changing any signal of level <= l.

The first step looks something like a proof of a non-interference condition as in the Goguen-Meseguer model, i.e. it shows that the inputs of entities of level not <=l can be deleted without effecting what is seen by entities of level <=l. What is the relationship between the state machine instantiation of the information model and the Goguen-Museguer non-interference model?

First of all, by "The Goguen-Meseguer model" we will hereinafter mean the model as set forth in [1]. We will only consider what are referred to as "static systems" in [1]. The first problem we encounter in comparing our state machine instantiation with the Goguen-Meseguer Model is that the Goguen-Meseguer model uses a different kind of automaton than the state machine instantiation.

The second problem we encounter is that the Goguen-Meseguer model allows us to express a much broader class of security policies that can be expressed in the state machine instantiation. In our reformulation of the state machine instantiation, we will broaden the policies expressible to include arbitrary non-interference assertions as in [1].

Before giving our reformulated instantiation, we will note a few assumptions about what users can "see" that seem to be implicit in the Goguen-Meseguer model. First, it seems implicit a user u cannot "see" the state machine making a transition from state s1 to state s2 if out(s1,u)=out(s2,u). In other words, a user cannot tell the difference between seeing a given output once and seeing it "twice in a row". If this were not the case, then whenever any user issued any command to the state machine, it would be seen by every user, either as a change in output or a "repeat" of the same output.

Second, it seems implicit that users cannot "see" time passing. In other words, a user cannot tell the difference between a sequence of state changes carried out over a "long" time and the same sequence of state changes carried out over a "short" time. Indeed, the Goguen-Meseguer model has no way of expressing this difference.

We now give a reformulation of our state machine instantiation in terms of Goguen-Meseguer-type state machines. Fix a state machine M consisting of a set of users, U, a set of states, S, a set of commands, C, a set of outputs, OUT, a function "out" from S x U into OUT, a function "do" from S x U x C into S and an initial state s. In addition, fix a set of commands A and sets of users G1 and G2. We wish to give an instantion of the information model to M whose information functions and information flow restrictions express the non-interference assertion A, G1 :| G2.

The set of possible worlds associated with M is the set of all finite sequences of elements of UxC. Since Goguen-Meseguer state machines are deterministic and have a unique starting state, the behavior of M during a given sequence of commands from users is completely determined by the sequence of users and commands issued. We choose finite sequences for the same reasons explained in subsection 5.2

We will now define a few functions we will need later. Given a possible world H=<(u[o],c[o]),...,(u[n],c[n])>, we can define a sequence of alternating states and elements of Ux6 ST(H)=<s[o], (u[o], c[o]), s[1], (u[1], c[1]), ..., s[n], (u[n], c[n]), s[n+1]> where s[o]= s (the initial state of M) and s[i+1]= do (s[i], u[i], c[i]) for i= , ..., u. In other words, ST(H) is just H with the states that M passes through M the course of H "interpolated".

Given a state s, we can define a functions V(s) from G2 into OUT by V(s)[g]=out(s,g) for all g in G2. V(s) is thus the "G2-tuple" of outputs "seen" by the member of G2 in state s.

We will now complete the instantiation. We associate two information functions with the non-interference assertion A,G1 :| G2 :

1. INPUT is the function which, given a possible world H, returns the subsequence of H consisting of the entries (u,c) where u is in G1 and c is in A.

2. VIEW is the function which, given a possible world H, returns a sequence obtained as follows:

   - Start with ST(H). Delete from ST(H) all entries (u,c) such that u is not in G2. Call the result X. (X will be a sequence of states and user-command pairs, not necessarily alternating).

   - Replace every entry s of X by V(s). Call the result Y. (Y will be a sequence of user-command pairs and functions from G2 into OUT).

   - Remove all consecutive repetitions of functions from G2 into OUT from Y. the result is VIEW(H).

INPUT simply extracts from H the history of inputs from users in G1 which are in the command set A. VIEW is slightly more complicated. Given H, it returns the history of commands from users in G2 and outputs to users in G2, with repeated outputs ignored. These functions are similar to the hidden-from and view functions of the first instantiation.

Finally, we specify that it is illegal for information to flow from INPUT to VIEW.

What is the relationship between the two definitions of security for M? First of all, a bit of pathology arises if A is non-empty and G1 and G2 overlap. Since users in G2 "know" what commands they're given, if a user u who is both G1 and G2 issues a command from A, then the users of G2 can deduce something about commands in A issued by users in G1. On the other hand, it is possible for A, G1 :| G2 to hold. For example, suppose G1=G2={u} and A={c} where do (s,u,c)=x for all states S. Then A,G1 :| G2 holds, i.e. u literally cannot interfere with himself by issuing c because c never causes any state change and so never causes any change in the output seen by u. The state machine instantiation takes into account the fact that u "knows" more than just what he "sees"; u also "knows" what he "does".

Thus, in the degenerate case where A is non-empty and G1 and G2 overlap, non-interference does not imply no flow from INPUT to VIEW. However, we do have

Proposition 1: If G1 and G2 are disjoint and A,G1 :| G2, then there is no flow from INPUT to VIEW.

Let p be the function which, given a possible world H, returns H with all entries in G1xA deleted. Proposition 1 will follow from the following

Lemma 1: If G1 is disjoint from G2 and A,G1 :| G2, then for all possible worlds H, VIEW(H)=VIEW (p(H)).

Proof of Proposition 1 from Lemma 1: We need to show that for any A in image(INPUT) and any B in image(VIEW), there exists a possible world H such that INPUT(H)=A and VIEW(H)=B. B in image(VIEW)=> there exists HO such that B=VIEW(HO). A in image(INPUT)=> A is a sequence of elements of G1xA. Let H= p(HO)^A. H is a possible world. Clearly, INPUT(H)=A. By the Lemma, VIEW(H)= VIEW(p(H))= VIEW(p(p(HO)^A))= VIEW(p(p(HO)))= VIEW(p(HO))= VIEW(HO)=B.

///

Before giving the proof of Lemma 1, we will note a few relevant facts.

Suppose H is a possible world, u is a user and c is a command. What is VIEW(H^<(u,c)>)? To compute VIEW(H^<(u,c)>), we must first compute ST(H^<(u,c)>). If s is the last element of ST(H) (i.e., the state of the machine after "doing" it), then ST(H^<(u,c)>) is ST(H)^<(u,c), do(s,u,c)>. Next, we delete all user-command pairs with user not in G2. Let X be the result of performing the operation on ST(H^<(u,c)>) is :

X^<(u,c), do(s,u,c)> if u is in G2

X^<do(s,u,c)> if u is not in G2

Next, we apply V to all states in the sequence. Let Y be the result of performing this operation on X; Then the result of performing the operation on the sequence above is:

Y^<(u,c), V(do(s,u,c))> if u is in G2

Y^<V(do(s,u,c))> if u is not in G2

Note that the last entry of Y is V(s). The last step in constructing VIEW(H^<(u,c)>) is to eliminate consecutive repetitions of values of V. The result of doing this to Y is VIEW(H). Therefore, we have

Fact 1: VIEW(H^<(u ,c)>)=

VIEW(H)^<(u,c), V(do(s,u,c))>, if u is in G2

VIEW((H)^<V(do(s,u,c))>, if u is not in G2 and V(5) is not = V(do(s,u,c))

VIEW(H), otherwise.

As mentioned above, for any possible world H, the last element of ST(H) is the state of M after "doing" H. It is easily seen that the last element of VIEW(H) is therefore the function form G2 to OUT which maps each user in G2 to the output "seen" by that user after H is "done". We denote the last element of a sequence Q by last(Q). A straightforward translation of the definition of non-interference in [GM] yields

Fact 2: A,G1 :| G2 if and only if for all possible worlds H, last(VIEW(H))= last(VIEW(p(H))).

Proof of Lemma 1: The proof is by induction on the length of H.

The base case is H=<> (the empty sequence). Then p(H) = <> = H, so VIEW(H) = VIEW(p(H)).

We now do the inductive step. Assume H=HO^<(u,c)> and VIEW(HO)=VIEW(p(HO))

Case1: (u,c) is in G xA. Then p(H)=p(HO^<(u,c)>)=p(HO), so by Fact 2, last(VIEW(H))= last(VIEW(p(H)))= last(VIEW(p(HO)))= last(VIEW(HO)).

Since G1 and G2 are disjoint, u is not G2, so by Fact 1, VIEW(H)=VIEW(HO^<(u,c,)>)=

VIEW(HO)^<V(do(s,u,c))>, if V(s) is not = V(do(s,u,c))

VIEW(HO), otherwise.

But last(VIEW(HO)) is V(s), so the only way that last(VIEW(H)) can = last(VIEW(HO)) is if the second case above holds, so VIEW(H)= VIEW(HO). By the inductive hypothesis, VIEW(HO)= VIEW(p(HO)), and p(HO)= p(H), so VIEW(H)= VIEW(p(H)).

Case 2: (u,c) is not in G xA and u is not in G2. Then p(H)= p(HO^<(u,c)>)=p(HO)^<(u,c)>. By Fact 2, VIEW(H)=

VIEW(HO)^<V(do(s',u,c))>, if V(s') is not=V(do(s',u,c))

VIEW(HO), otherwise.

Where s=last(ST(HO)). Again by Fact 2, VIEW(p(H))=VIEW(p(HO)^ <(u,c)>)=

VIEW(p(HO))^<V(do(s',u,c))>, if V(s') is not= V(do(s',u,c))

VIEW(p(HO)), otherwise.

Where s'=last(ST(p(HO))). By the inductive hypothesis, VIEW(HO)= VIEW(p(HO)). Therefore, V(s)= last(VIEW(HO))= last(VIEW(p(HO)))= V(s').

Now, Suppose V(s)= V(d0(s,u,v)) but V(s') is not= V(do(s',u,c)). Then VIEW(H)= VIEW(HO) and VIEW(p(H))= VIEW(P(HO))^<V(do(s',u,c))>. By Fact 2, last (VIEW(H))= last(VIEW(p(H))). But then

V(s')=V(s)= last(VIEW(HO))= last(VIEW(H))= last(VIEW(p(H)))= V(do(s',u,c)), a contradiction. Therefore, if V(s)= V(do(s,u,c)), then V(s')= V(do(s',u,c)), and so VIEW(H)= VIEW(HO)= View(p(HO))= VIEW(p(H)).

Next, suppose V(s) is not= V(do(s,u,c)) but V(s')= V(do(s',u,c)). By an argument completely analogous to that of the previous paragraph, this lead to a contradiction, so if V(s) is not= V(do(s,u,c)) then V(s') is not= V(do(s',u,c)); in this case,

VIEW(H)=VIEW(HO)^<V(do(s,u,c))>

VIEW(p(H))=VIEW(p(HO))^<V(do(s',u,c})>

VIEW(HO)= VIEW(p(HO)) by the inductive hypothesis, and by Fact 2, V(do(s,u,c))=last(VIEW(H))= last(VIEW(p(H)))= V(do(s',u,c)), so VIEW(H)= VIEW(p(H)).

Case 3: (u,c) is not in G xA and u is in G2. Then p(H)= p(HO)^<(u,c)>. By Fact 1,

VIEW(H)=VIEW(HO)^<(u,c), V(do(s,u,c))>

VIEW(p(H))=VIEW(p(HO))^<(u,c), V(do(s',u,c))>

VIEW(HO)= VIEW(p(HO)) by the inductive hypothesis, and the last elements of the above sequences are= by Fact 2, so again, VIEW(H)= VIEW(p(H))

///

The converse of Proposition1 fails in a non-pathological case however.

Proposition 2: No flow from INPUT to VIEW does not imply A,G :| G .

Proof: Consider the following state machine:

U={u1,u2,u3}

S={0,1}x{0,1}

C={flip1,flip2}

OUT={0,1}

out((b1,b2),u)=

    b1 if u=u1
    0 otherwise

do ((b1,b2),u,c)=

```
(b1,b2)     if u=u1
(b1,b2)     if u=u2 and b2=0
(b1,b2)     if u=us and b2=1 and c=flip2
(1-b1,b2)   if u=u2 and b2=1 and c=flip1
(1-b1,b2)   if u=u3 and c=flip1
(b1,1-b2)   if u=u3 and c=flip2
```

so=(1,1)

Briefly, the state consists of 2 flags. u1 can see the first flag, while u2 and u3 can't see anything. There are 2 commands, one to flip the first flag and one to flip the second flag. Commands from u1 are always ignored. Commands from u2 to flip the second flag are always ignored, whereas commands form u2 to flip the first flag are carried out if the second flag is 1, and are ignored otherwise. Commands from u3 to flip either flag are always carried out.

Let A={flip1}, G1={u2} and G2={u1}.

Claim 1: A,G1 :| G2 does not hold. Consider the possible world H=<(u2,flip1)>. After H, u1 is "seeing" a p(H)=<>/ After p(H), u1 is "seeing" a 1. By definition of non-interference, the above assertion does not hold.

Claim 2: There is no flow from INPUT to VIEW. Rather than give a completely rigorous proof, we will simply indicate why claim 2 is true. We wish to show that u1 cannot possibly deduce anything about u2's inputs by observing his own inputs and his outputs. The reason this is true is because anything that u1 sees could be the result of u3 issuing flip2, u3 issuing flip1 a certain number of times, and u2 issuing any sequence of commands. In other words, no matter how many times u1 sees the first flag flip, it could always be the result of u3 issuing flip 1, and if u3 in addition issues flip 2 immediately, all of u2's inputs are "masked out".

The essential difference in this example between the Goguen-Meseguer model and the information model instantiation, is that Goguen-Meseguer requires that there be no change in what u1 sees when u2's inputs are deleted <u>while holding u3's inputs fixed</u>. The Goguen-Meseguer model requires u3's inputs to be held fixed even though u1 has no way of knowing what they are.

In conclusion, the state machine instantiation of the information model given above seems (if certain pathological situations are ruled out) to be a generalization of the Goguen-Meseguer model. It is a proper generalization in the sense that it is implied by Goguen-Meseguer but does not always imply Goguen-Meseguer.

REFERENCES

[1] Goguen, J.A. and Meseguer, J., <u>Security Policies and Security Models</u>, Proceedings of the 1982 Symposium on Security and Privace, April 1982.

A SEMANTICS OF READ

Leo Marcus and Tim Redmond[1]
Computer Science Laboratory
The Aerospace Corporation
P.O. Box 92957
Los Angeles, CA 90009

ABSTRACT

We give a rigorous mathematical definition for the concept of a variable being read by a process and give evidence that this definition captures the correct intuitive notion with applications to security specification and verification. We examine the expressibility of "read" in various temporal logics, getting some positive and some negative results. We define what it means for a host machine to implement a target machine in such a way that a given variable is not read. We show that "read" is theoretically decidable and give some proof rules. In a later paper we intend to give axioms and proof rules for such "protected implementation" proofs in the context of the State Delta Verification System being developed at The Aerospace Corporation (see[1]).

This formulation is rather new, and we are still exploring its technical properties and applications.

[1]This work was supported in part by the Aerospace Sponsored Research program.

INTRODUCTION

In dealing with computer security, a major consideration is protecting certain registers from being read from or written into by unauthorized processors or users. It is fairly trivial to define the semantics of write: if the value of the register changes, it was written into. (This is ignoring the rare occasion when writing a value which is the same as the current value may be of interest.)

However, detecting when a register's value has been read is a much more delicate matter. For a register x to be read, we do not require that the reader actually "look at" or access the x, nor that the reader learn the value of the contents of x in any way. We view "reading" as a special case of the general problem of information flow. Intuitively, we will consider a register x to have been read by (or during) process P, if some non-public or protected information about its contents becomes known during an execution of P, i.e., if the behavior of P is dependent on the value of x in some specifiable or observable way. This means that the concept of "non-public" must be made explicit in every specific case of read.

A superficial approach to the semantics of read yields the following examples. If the right hand side of an assignment statement consists of the program variable x by itself, then x is read. If the expression x − x (subtraction) is on the right hand side, it is not completely clear whether we want to consider x to have been read or not. If x appears in a condition for a branch, where the outcome of branch depends on the value of x, we probably do want to consider that x has been read, even if we don't need to know its explicit value.

These examples all rely on the presence of some syntax for their formulation. The situation is clearly different in the case of security verification. We shall consider a prototypical relation between an adversary, A, and a process, P. The adversary tries to learn something new about x by examining the behavior of the process and by using the public knowledge, K, available about x. A, in general, does not have complete knowledge about the syntax of P; A may observe P in operation or may wait until P has terminated (if ever), and then analyze the results to deduce the new information about x. In this case, disclosure of new knowledge about x means that some behavior of x which is *a priori* possible in the context of of K, is discovered to be impossible in light of P.

The link between the syntactic and the general formulation can be seen, for example, in the simple assignment statement above: a possible value of x (actually, all but one possible value of x) is eliminated by examining the value of the left hand side. Likewise in the branch example, the negation of the realized branch predicate is discovered to be impossible at that point in time in that computation (the realized branch predicate is discovered to be true).

Our approach elevates this "public knowledge" to a position of prominence in the definition. K plays a dual role in the sense that it can be used by the adversary to deduce information about x based on observation of P, but K also is the criterion for deciding if that information about x is new. For example, if the public knowledge about x is weak (e.g., K = TRUE), and x is not an explicit variable of P, then x is not read by P because there is no way to connect x with the behavior of P. Also, if K is too strong (e.g., K = FALSE), then x is not read by P, because P could not possibly increase our knowledge about x. Actually, x can alternate between being read and not being read with respect to K as K increases in strength. On the other hand, for any non-trivial process P and register x, there is some public knowledge such that with respect to it, P reads x: simply take K to be x = v for some variable, v, of P.

We assume a strict distinction on the one hand between the variables <u>of</u> a process, Var(P), which P knows everything about at all points of all computations and by definition reads, analogous to real locations in a machine for P, and on the other hand, external variables which P may or may not read, depending on K.

We take the view that the specification of a process is a way of <u>restricting</u> the possible computations that the user can perform, rather than <u>permitting</u> them. We are interested in protecting against the inadvertent read, not finding which registers are always read. Thus, the fewer computations the user can perform (the fewer models the process has), the less chance of x being read.

There are four separable concerns which need formalization: the new information learned about x by P is "information", it is "new", it is "about" x, and it is learned "by" P. As mentioned above, the "newness" will be measured in relation to K; "information" is taken to be a set of possible computations. The "about x" and "by P" are handled by looking at the restriction of a model of K to x, and combining this with a computation of P.

To formalize the above discussion in precise mathematics, we utilize the concepts of model theory, in the sense of[2]. We assume only very basic acquaintance with logical concepts and model theory. We start out with a model (computation) of K, and we restrict it to x (ignore the other variables). This represents a possible behavior of (or information about) x consistent with K. Now take a model (computation) of P and see if we can superimpose the above restricted model onto this model of P in a way which is consistent with K, i.e, such that the combination is a model of K. If we cannot, then we have learned that this behavior of x is ruled out by this particular computation of P, and x is read. It could be that the forbidden behavior (the information) is specifiable by a sentence in a given language. This means that there is a sentence, F, such that the above holds for all models of $K \wedge \neg F$. (In other cases, it may be that a particular model or set of models is ruled out, but this model or set of models is not specifiable in the language.) If **M** does not read x with respect to K, then the adversary cannot deduce anything about x that does not already follow from K. See Lemma 3.

We examine the possibility of expressing the necessary semantics within various temporal logics and come to the conclusion that this is impossible in some cases.

There are several possible variations of the formalization which seem reasonable. An important task is to examine examples and results about their interdependencies in order to determine which ones correctly represent our intuition.

See[3],[4], and[5] for discussions of similar problems.

EXAMPLES

Example 1: Consider a system with two processes sharing a common CPU: P_1, which uses variable y, and P_2, which uses variable x. Each has exclusive use of the CPU for five clock cycles. Every five clock cycles the CPU will swap the other process in if it so requests, and the first process is then swapped out. Let K be the description of this system. It is true that x is read by P_1 with respect to K, since when P_1 is running, it knows that x cannot be changing.

However, the only information that P_1 gains about x is that x is not active when P_1 is running. Assume now that the system contains another variable, "status", that holds the name of the active process for each point in the timeline, and let K' be the description of the new system along with a particular choice of values for status. Then P_1 does not read x with respect to K'.

For a more detailed analysis of this example see Example 4.

Example 2: If K is $x = 0$ or K is TRUE, then no process reads x with respect to K. K either contains all the possible information about x, or does not give any connection between x and other variables. If K is $v = 0 \rightarrow x = 0$, then any computation satisfying $\exists t(v(t)=0)$ reads x with respect to K. Notice that this is an example of $K_1 \rightarrow K_2 \rightarrow K_3$ and a computation that reads x with respect to the middle K but not with respect to the two outer ones.

If K is '$0<x<5 \wedge 0<v<5 \wedge x=v$' or '$0<x<5 \wedge 0<v<5 \vee x=v=6$' then x is read by every process with respect to K. If K is just '$0<x<5 \wedge 0<v<5$', then x is not read by any process with respect to K. This is an example of $K_1 \rightarrow K_2 \rightarrow K_3$ such that x is read with respect to the outer ones, but not read with respect to the middle one.

Example 3: If K = '$u + x = v$', and u and x are not variables of the process P, then P does not read x with respect to K.

THE FORMAL DEFINITION OF READ

We consider a computation for a process to be specified when we know the values of all process variables at every point in time. A process thus determines a set of computations, its set of possible computations. Formally, this becomes a class of models based on an arbitrary linear order $\langle T, <\rangle$ representing the timeline of the computation. At each point t of T a state is specified by defining the values of the functions v(t), where v can be thought of as a program variable and v(t) its contents at t.

For a given process, T may change from computation to computation, but the domain of values is fixed.

> **Definition 1:** A computational model for program variables V over the linear order $\langle T, <\rangle$ is a structure of the form $M = \langle T \cup D; <, v, \ldots, R, \ldots\rangle_{v \in V}$ where v, ... are function symbols to be interpreted as functions from T to D (the "domain of data") and R,... are relations (and functions) on D.

Let $L(M) = \{<\} \cup V \cup \{R, \ldots\}$ be the language of **M**, $Var(M) = V$ be the set of variables of **M**.

A process (or program or theory) σ is a class of computational models with V, D, and R, ... fixed. We call $\langle D; R, \ldots\rangle$ the domain of σ. In that case we write $M \models \sigma$ if $M \in \sigma$ and define $L(\sigma) = L(M)$ and $Var(\sigma) = Var(M)$. If ϕ is a sentence in the appropriate language, then $M \models \phi$ has the standard meaning (see[2]).

So for example, by using time as an explicit variable we can say

$$M \models (\exists t_1 \in T)(\forall t_2 \in T)(t_1 = t_2 \vee t_1 > t_2), \text{ or}$$

$$M \models \exists t_1 \in T)(\forall t_2 \in T)(t_2 \geq t_1 \rightarrow \wedge_{v \in V} v(t_2) = v(t_1))$$

(two versions of terminating computation), or $M \models \forall t \in T(v(t) > u(t))$, etc.

In the following, when we say "model" we mean "computational model".

Now we want to define the formal counterpart of the intuition of comparing a possible behavior of x to actual behaviors of x allowed by **M**. Remember, our definition will say that x is read by **M** with respect to K if there is a behavior of x that is possible according to K, but which is not allowed by **M**.

Informally still, given a model **M** with x as a variable, the behavior of x in **M** is simply the model obtained by ignoring all the other program variables. Formally, this is the restriction of **M** to x.

> **Definition 2:** If **M** is the above model ($Var(M = V)$ and U is any set of program variables (typically, but not necessarily, $U \subseteq V$) we define the restriction of **M** to U by $M \restriction U = \langle T \cup D; <, v, \ldots, R, \ldots\rangle_{v \in U \cap V}$. **M** is an expansion of $M \restriction U$.

In compliance with the standard terminology, we use "expansion" for a model obtained by enriching the language, and "extension" for a model obtained by enlarging the underlying set.

A behavior of x which is possible according to K is just $M_1 \restriction \{x\}$ for $M_1 \models K$. The behavior of x is allowed by a model **M** if when we glue this behavior to **M** we get a model of K.

If $M_1 = \langle T \cup D; <, v, \ldots, R, \ldots\rangle_{v \in V_1}$ and $M_2 = \langle T \cup D; <, v, \ldots, R, \ldots\rangle_{v \in V_2}$ are models over the same timeline and with the same domain, $M_1 \restriction V_1 \cap V_2 \cong M_2 \restriction V_1 \cap V_2$, then $M_1 \cup M_2 = \langle T \cup D; <, v, \ldots, R, \ldots\rangle_{v \in V_1 \cup V_2}$.

Here is the main definition:

> **Definition 3:** Let **M** be a model over T as above, $K(\bar{v}, x)$ a sentence in $L(M) \cup \{x\}$. **M** reads x with respect to K if **M** has an expansion to a model of K but there is $M_1 \models K$ over T such that $(M_1 \restriction \{x\}) \cup M \not\models K$.

If $K = K(\bar{v}, \bar{u}, x)$ contains additional variables $\notin L(M)$ besides x, then the definition becomes:

Definition 4: M (over T) reads x with respect to K($\bar{v}$, $\bar{u}$, x) if **M** has an expansion to a model of K, but there is a model $\mathbf{M}_1 \models K$ over T such that $\mathbf{M}_1 \lceil \{x\} \cup \mathbf{M}$ has no expansion to a model of K.

The intuitive reason that we insist that **M** be consistent with K is that otherwise, the "adversary" would not be able to observe **M** at all; K and **M** could not coexist. The "new information" about x is that $M_1 \lceil \{x\}$ is not allowed. Thus, the possibilities for x satisfying K are restricted.

Definition 5: x is read by σ with respect to K if there exists $M \models \sigma$ such that x is read by **M** with respect to K.

Example 4: Now let us return to Example 1. Let **M** be any model of P_1; remember, **M** does not mention x. **M** is obviously consistent with K, i.e., it has an expansion to a model of K. For example, x can be defined to be always idle. If y is not idle in **M**, then **M** reads x with respect to K, since there is a model M_1 of K which has x changing exactly at the time that **M** has y changing.

Now consider the case with the added status word. Let M_0 be a particular graph of the status word and let K' = K $\cap$ {M: $M\lceil$ {status} = M_0}. Then x is not read by any **M** with respect to K', since the non-active/potentially active behavior of x is determined by status. $\dashv$

<u>THE PADDED VERSION OF READ</u>

Now we shall give an alternate definition which relaxes the condition that the behavior of x which is excluded by **M** must be realized over a timeline <u>identical</u> to M's. We shall be looking at models of K whose timelines can be <u>superimposed</u> on the timeline of M in a consistent manner. This is the concept of "padding".

We shall not give a formal definition of "padding", but it is sufficient to think of a padded version of M as a model over a larger timeline in which some states of M are spread out (in either direction).

Definition 6: x is strongly read by M over T with respect to K if there is a padding of **M** to a model of K($\bar{v}$, x), but there is $M_1 \models K$ over $T_1 \supseteq T$ and M_2, a padding of M to T_1, such that $M_1 \lceil \{x\} \cup M_2 \not\models \{K\}$.

The intuition is that we have some property of x consistent with K: this is embodied in our choice of M_1. However, by "running" σ under certain circumstances (over the "T_1-pad" of M) we eventually come to the conclusion that this property of x consistent with K cannot be true.

Definition 7: x is strongly read by σ with respect to K if there is $M \models \sigma$ such that x is strongly read by M with respect to K.

Definition 8: A language L is <u>paddable</u> if padding preserves L-equivalence between L-models.

Theorem 1: In the case of paddable language, if x is strongly read by σ with respect to K, then x is read by σ with respect to K.

For example,

Theorem 2: The language of "weak until" (WU) is paddable, where

Definition 9: WU is the set of sentences formed from first order logic (not containing <) and closed under conjunction, disjunction, negation, and U*:

$(M, t_0) \models PU^*Q$ iff

$(\exists t_2 \geq t_0)\ (Q(t_2) \wedge (\forall t_1)(t_0 \leq t_1 \leq t_2 \rightarrow P(t_1)))$.

THE CASE OF SPECIFIABLE INFORMATION

So far, we have studied "read" in a general framework, where the information learned about x took the form of a given model or behavior which was excluded from the set of possible behaviors. Now we examine the situation in which the information read about x is expressible in a given language.

The reason for this new definition is that the previous definition was in a sense too strong, i.e., x could be read, but the behavior which was excluded was not specifiable, and thus the information gained could not be "used".

Definition 10: M (over T) discovers $F(\bar{v}, x)$ with respect to K if

1. there is an expansion of M to a model of K,
2. there is $M_1 \models K \wedge \neg F$ (F does not follow from K),
3. if $M_1 \models K \wedge \neg F$, then $M_1 \lceil \{x\} \cup M \models \neg K$.

For example, if K is $v(t)=0 \rightarrow x(t)=0$, then M satisfying $v(t)=0$ discovers $x(t)=0$.

Discovery satisfies the following intuitive properties:

Lemma 3:

1. If M discovers F with respect to K, then M does not discover $\neg F$ with respect to K.
2. If M discovers F_i with respect to K, then M discovers $F_1 \wedge F_2$ with respect to K.
3. If M discovers $F(\bar{v}, x)$ with respect to K, then M reads x with respect to K.

Actually this definition works just as well for F an arbitrary set of models.

READING DURING AN INTERVAL

We can further refine the main definition to talk about <u>when</u> the reading takes place.

Definition 11: M (over T) reads x with respect to K <u>during</u> the interval $I \subseteq T$ if M has an expansion to a model of K, but there is $M_1 \models K$ over T such that $M_1 \lceil I$ cannot be extended to $M_2 \models K$ over T such that $M_2 \lceil \{x\} \cup M \models K$.

Some intuitively desirable properties hold:

Lemma 4:

1. "M (over T) reads x with respect to K during the interval T" (M's whole timeline) reduces to the original definition of M (strongly) reads x with respect to K.
2. "M (over T) reads x with respect to K during the interval I" is not the same as "$M \lceil I$ reads x with respect to K."
3. If M does not read x with respect to K during the interval I, then the same holds true for every subinterval of I.

Unfortunately,

Example 5: Read protection during intervals does not finitely compose. That is, there is M over T, $t_1 \leq t_2 \leq t_3 \in T$, K, such that M does not read x with respect to K during $[t_1, t_2]$, M does not read x with respect to K during $[t_2, t_3]$, but M does read x with respect to K during $[t_1, t_3]$.

Read protection during intervals does not compose in the limit, either. That is, there is M over $T = \cup_{i<\omega}[t_0, t_i]$ such that x is not read with respect to K during $[t_0, t_i]$ for all i, but M does read x with respect to K during T. ⊣

READ AND RELEVANCY

Definition 12: $\bar{v}$ reads $\bar{x}$ with respect to $K(\bar{v}, \bar{u}, \bar{x})$ if there is a model M such that $M\lceil\{\bar{v}\}$ and $M\lceil\{\bar{x}\}$ have expansions to models of K, but

$M\lceil\{\bar{v}\} \cup M\lceil\{\bar{x}\}$ does not.

Note that if M reads x with respect to K, $v \in L(M)$, then v reads x with respect to K.

Definition 13: $\bar{v}$ reads $\bar{x}$ with respect to K and $\bar{y}$ if there is M such that $M\lceil\bar{x}\cup\bar{y}$ and $M\lceil\bar{v}\cup\bar{y}$ have expansions to models of K, but $M\lceil\bar{v}\cup\bar{x}\cup\bar{y}$ does not.

Note that $\bar{v}$ reads $\bar{x}$ with re ıect to K and $\varnothing$ iff $\bar{v}$ reads $\bar{x}$ with respect to K.

<u>Notation</u> $I_K(\bar{x}, \bar{u}, \bar{v})$ holds if $\bar{v}$ does not read $\bar{x}$ with respect to K and $\bar{u}$. I_K is to be compared to the "irrelevancy relation" from Pearl and Paz[6]. There the common knowledge K is not made explicit.

Lemma 5:

1. $I_K(\bar{x}, \bar{u}, \bar{v})$ iff $I_K(\bar{v}, \bar{u}, \bar{x})$
2. $I_K(\bar{x}, \bar{u}, \bar{v} \cup \bar{w}) \rightarrow I_K(\bar{x}, \bar{u}, \bar{v}) \wedge I_K(\bar{x}, \bar{u}, \bar{w})$.
3. $I_K(\bar{x}, \bar{u}, \bar{v} \cup \bar{w}) \rightarrow I_K(\bar{x}, \bar{u} \cup \bar{w}, \bar{v})$.
4. $I_K(\bar{x}, \bar{u}, \bar{w}) \wedge I_K(\bar{x}, \bar{u} \cup \bar{w}, \bar{v}) \rightarrow I_K(\bar{x}, \bar{u}, \bar{v} \cup \bar{w})$.

These are conditions (11.a), (11.b), (11.d), and (25) of ([6]).

Condition (11.c) from[6] is not true: i.e.,

$$I_K(\bar{x}, \bar{u} \cup \bar{v}, \bar{w}) \wedge I_K(\bar{x}, \bar{u} \cup \bar{w}, \bar{v}) \not\rightarrow I_K(\bar{x}, \bar{u}, \bar{v} \cup \bar{w}).$$

EXPRESSIBILITY OF READ

Definition 14: A logic L expresses read if for all $K(\bar{v}, \bar{u}, x) \in L$, there is $\sigma \in L$ such that M does not read x with respect to K if and only if $M \models \sigma$.

Note that it in order to show that L does not express read, it is sufficient to find M_1, M_2 satisfying the same L-sentences such that one reads and the other does not.

This method can be used to prove:

Theorem 6: WU doe~ ¬ot express read.

For similar results, see[7]. The proof we have involves non-standard timelines (i.e., not isomorphic to the natural numbers) and infinite domain set. It is not known if all WU-equivalent models over timeline ω with finite domain set are also read-equivalent.

One positive result though:

Lemma 7: If K is static (over one-point timelines) or time universal, then read with respect to K is expressible.

READ AS A GENERALIZED QUANTIFIER

In this section we express "read" as a generalized quantifier in the sense of[8]. This gives us a syntax to use in reasoning about read statements.

We introduce the quantifier $\mathfrak{R}_x$ where $M \models \mathfrak{R}_x K$ will mean that M reads x with respect to K (when K does not contain any occurrences of $\mathfrak{R}_x$.)

Example 6:

1. The example of K from Example 1 generalizes to $\models \exists y \neg G(y) \wedge G(v) \rightarrow \mathfrak{R}_x(G(v) \rightarrow \neg G(x)))$ (which is a special case of 4 below.)
2. The example of K being '$0<x<5 \wedge 0<v<5 \wedge x=v$' from Example 2 generalizes to $\models \exists^{\geq 2} y G(y) \wedge G(v) \rightarrow \mathfrak{R}_x(G(v) \wedge x = v)$.
3. The example of '$0<x<5 \wedge 0<v<5 \vee x=v=6$' generalizes to $\models \exists y G(y) \wedge \neg G(v) \rightarrow \mathfrak{R}_x(G(v) \vee x = v)$.
4. The example of '$v=0 \rightarrow x=0$' generalizes to $\models G_1(v)$

$\wedge \exists y G_2(y) \wedge \exists y \neg G_2(y) \rightarrow \Re_x(G_1(v) \rightarrow G_2(x))$. ⊣

Theorem 8: The following are valid statements (true in every L-model, where $x \notin L$):

1. If x does not occur in G_1 and v does not occur in G_2, then $\neg\Re_x(G_1(v) \wedge G_2(x))$.
2. If x does not occur in G_1, then $\Re_x(G_1 \wedge G_2) \rightarrow G_1 \wedge \Re_x G_2$.
3. $\Re_x(G_1 \wedge G_2) \rightarrow \Re_x G_1 \vee \Re_x G_2$.
4. $\exists y G_1(y) \wedge \exists y \neg G_1(y) \wedge \exists y G_2(y) \wedge \exists y \neg G_2(y) \rightarrow \Re_x(G_1(v) \leftrightarrow G_2(x))$.
5. $\forall y G(v, y) \rightarrow \neg\Re_x G(v, x)$. (This implies $G_1(v) \rightarrow \neg\Re_x(G_1(v) \vee G_2)$).
6. $\forall y \neg G(v, y) \rightarrow \neg\Re_x G(v, x)$.
7. $G_1(v) \wedge \exists y G_2(y) \wedge \exists y \neg G_2(y) \rightarrow \Re_x(G_1(v) \rightarrow G_2(x))$.

We do not know how to expand the above set to get a complete axiomatization of read.

Note that the following is not valid:
$F \wedge \Re_x K \rightarrow \Re_x(K \wedge F)$, for x not occurring in F.
This is the converse of 2 above.

Example 7: There are G_i such that the following are satisfiable:

1. $\Re_x G_1 \wedge \Re_x(\neg G_1)$
2. $\Re_x G_2 \wedge \neg\Re_x(\neg G_2)$
3. $\neg\Re_x G_3 \wedge \neg\Re_x(\neg G_3)$

Take G_1 = x>v, $G_2 = \neg(P(x) \wedge Q(v))$, and G_3 = TRUE (or FALSE).

DECIDABILITY OF READ

In this section we show that "read" is decidable. More precisely, we show that for a fixed (finite) domain D, the theory consisting of the set of sentences in the logic with $\Re_x$ which are true in all computational models over D with countable timelines is decidable. The result follows by interpreting $\Re_x$ in the second order monadic theory of countable chains, shown to be decidable in[9]. Second order monadic logic allows quantification over subsets, but not arbitrary relations or functions. This result is, of course, only of theoretical interest without practical application, since the size of the domain is typically very large.

Let D = $\{d_1, ..., d_n\}$ be a given finite domain. Let $\Re_x$ be the read quantifier for countable computational models over D. That is, $M \models \Re_x K$ if M is a computational model over D with countable timeline T, there is an expansion of M satisfying K, and there is $M_1 \models K$ such that M_1's timeline is T and $M \cup M_1\lceil\{x\} \not\models K$. Let Th($\Re_x$) be the set of sentences in the above language which are satisfied in every computational model over D with countable timeline.

Theorem 9: Th($\Re_x$) is decidable.

Proof: In order to interpret Th($\Re$)$_x$) in the monadic second-order theory of linear order, first we interpret $\Re_x$ in the logic allowing quantification over functions from T to D:

$M \models \Re_x K$ iff

$M \models (\exists f_x{:}T \rightarrow D)\, K(v^M, f_x) \wedge (\exists f_v', f_x', ... {:}T \rightarrow D)[K(f_v', f_x') \wedge \neg K(v^M, f_x')]$.

Now we write $\exists f_x$ as a finite sequence of set quantifiers over T:

$(\exists f_x{:}T\rightarrow D)\phi(f_x) \leftrightarrow (\exists D_1, .., D_n \subseteq T)$ (the D_i are pairwise disjoint $\wedge\ \phi^*(D_1, ..., D_n))$, where ϕ^* replaces $...f_x(t)...$ with $\wedge(D_i(t) \rightarrow ...d_i...)$. ⊣

PRACTICAL FORMULATION OF READ PROTECTION

The real-world situation that served as the motivation for the development of this concept of read, and that will serve as the final judge of its viability, is the following: given a host machine (code), H, and a target machine (specification), T, H implements T in such a way that variable x is not read. See[1] or[10].

In order to make this statement precisely fit the mold of the formal definition of read presented in this paper, we need to specify what formulas (programs) will play the roles of σ and K, the relevant process and public knowledge.

Proposal:

Definition 1: "H implements T in such a way that x is not read" if

1. H implements T
2. T does not read x with respect to H.

So we are suggesting that $\sigma = T$ and $K = H$. x can be (and usually is) an explicit variable of H, but not of T. T is the process whose behavior the adversary may examine in order to try to learn some new information about x, and the (structure of the) host machine H is considered to be public knowledge.

The variables of T must be considered to be associated already by the implementation mapping to relevant variables in H for this formulation to make sense. (σ and K typically have variables in common.) The role x plays in H is (hopefully) hidden from T; T exists at a level of abstraction (tailored, perhaps, to suit the adversary's read rights) so that the behavior of x cannot be inferred from the behavior of T, even taking into account the structure of H.

Of course, it could be that different mappings which both give correct implementations would give different results for the security questions. Given a host and target, a "security analysis" would consist of characterizing those mappings with respect to which H implements T in such a way that x is not read.

ACKNOWLEDGEMENT

We thank Sue Landauer for her careful reading of the manuscript, her suggestions, and her corrections.

References

1. L. Marcus, S. D. Crocker, and J. R. Landauer, "SDVS: A System for Verifying Microcode Correctness", *17th Microprogramming Workshop*, IEEE, October 1984, pp. 246-255.

2. C. C. Chang and H. J. Keisler, *Model Theory (Second Edition)*, North-Holland. 1977.

3. J. Halpern and M. O. Rabin, "A Logic to Reason about Likelihood", *Fifteenth Annual ACM Symposium on Theory of Computing*, ACM, 1983, pp. 310-319.

4. S. Goldwasser, S. Micali, C. Rackoff, "The Knowledge Complexity of Interactive Proof Systems", *1984 ACM Foundations of Computer Science*, ACM, 1985, pp. 291-304.

5. V. Nguyen and K. Perry, "Do We Really Know What Knowledge Is?", Tech. report RC 11830, IBM T. J. Watson Research Center, April 1986.

6. J. Pearl and A. Paz, "GRAPHOIDS:A Graph-based Logic for Reasoning about Relevance Relations", Tech. report R-53-L-1, CSD-850038, Computer Science Department, UCLA, April 1986.

7. L. Marcus, T. Redmond, and S. Shelah, "Completeness of State Deltas", Tech. report ATR-85(8354)-5, The Aerospace Corporation, 1985.

8. J. Barwise and S. Feferman, *Model-Theoretic Logics*, Springer-Verlag, 1985.

9. Michael O. Rabin, "Decidability of second-order theories and automata on infinite trees", *Transactions of the American Mathematical Society*, Vol. 141, 1969, pp. 1-35.

10. Leo Marcus, "Implementation Mapping between Programs", Tech. report ATR-84(8478)-3, The Aerospace Corporation, 1984.

A STANDARD NOTATION IN COMPUTER SECURITY MODELS

O. Sami Saydjari
Timothy Kremann

National Computer Security Center
Attn: Office of Research and Development, C3
9800 Savage Road
Fort George G. Meade, MD 20755-6000
(301) 85'-4488

ABSTRACT

In the burgeoning field of computer security, there has been a lack of standard notation for representing models. This paper introduces such a notation, called Set Normal Form (SNF), based on set theory. The paper recasts the Bell and LaPadula, Biba Integrity, Role Enforcement, and Multilevel Object models in this notation. The standardization should facilitate the comparison of models in terms of security, completeness and level of abstraction.

INTRODUCTION

The purpose of this paper is not to present new research in computer security. Instead, its intent is to offer a standard notation based on set theory. The objectives are to provide a common language for model expression and facilitate the comparison of models.

Historically, a rather difficult mixed notation was adopted because of the significant impact of the Bell and LaPadula (BLP) model. The set theory notation, however, seems to be more understandable and flexible. A set theory casting of a simplified form of the BLP model[1], the dual Biba Integrity Model[2], a role enforcement model[3], and a Multilevel Object (MLO) model[4] are provided as appendices and discussed in detail in this paper. We shall call this notational representation "Set Normal Form (SNF)."

The essence of this paper is in the appendices. There is a discussion section for each of the four appendices corresponding to the four models rewritten in SNF. The purpose of each discussion section is to provide: (1) highlights, (2) clarifications, (3) motivations, and (4) explanations of any deviations from the original model.

The SNF castings of the four models chosen are intended to capture the basic essence of each model. Many of the more subtle features have been intentionally left out for simplicity. The point of these representations is only to show basic examples of how SNF is applied.

DISCUSSION

Notation Primitives

Notation primitives are descriptions of notions that are either already well-defined in computer science or common to many of the different models. The motivation behind introducing these primitives is to factor out common characteristics of sets introduced in this paper. We explain the primitives once and then use them to provide a shorthand method of referring to the characteristics.

The notion of "maps-completely-to" is introduced. Set A is said to map-completely-to set B if every element in set A maps to some element in set B. For example, requiring every object in a system to have an associated security label is the same as the set of objects maps completely to the set of classification labels.

The notion of "maps-uniquely-to" is also introduced. Set A maps-uniquely-to set B if no element in A is mapped to more than one element from B. For example, an object must have only one classification associated with it at any time (that is, it cannot be both TOP SECRET and UNCLASSIFIED at the same time).

The "maps-uniquely-to" and "maps-completely-to" primitives are combined and called "maps-fully-to." All classification labeling of objects and subjects must conform to this primitive. In other words, all objects and subjects must have one, and only one, classification level associated with them.

The power set primitive, PS, is introduced to indicate all of the possible subsets of a given set. PS is not explicitly used in the representation of models in this paper, but the need for it is anticipated for richer descriptions.

The "is-a-hierarchy-on" primitive (used in the MLO Model) describes the relationship between containers and atoms. Containers are objects which have descriptors of other objects while atoms are self-contained objects. A hierarchy, as defined in appendix A, is used to show the container-atom relationship between objects.

The requirements placed upon this hierarchy are that its digraph representation contain no cycles. The model requires that each container's security level dominates the level of each entity that it contains. The actual hierarchy, as defined here, is a collection of acyclic rooted digraphs which may overlap.

In this hierarchy we require that no container be empty. This means that all leaf nodes are atoms and all nonleaf nodes are containers. Any empty containers will contain a special leaf node called the null atom. This is an arbitrary restriction which simplifies the MLO SNF. The "is-a-leaf-in" indicates that the corresponding digraph has no outgoing arcs from this node.

Finally, the concept of "is-a-partial-ordering" defines the standard mathematical concept of a partial ordering on a set. This primitive is very important in establishing order in a normally unordered set. In particular it allows the definition of dominance for classification levels.

Another concept introduced in SNF is set dependence. The motivation is based on a confusion about which sets are specified by the user (or security administrator or system designer) and which sets are constrained by the choice of other sets. A level-one set is one which you have complete freedom to chose. A level-two set is one that depends on, or is constrained by a level-one set. Similarly, several levels of dependence can be traced in most models.

<u>Bell and LaPadula Model</u>

The level-one sets of the BLP model are the compromise levels (C_L), objects (O), and subjects (S), and the access rights (A) that will be used to restrict information flow.

The set of compromise levels is simply an unordered enumeration of the classification labels that exist in the system-for example, {SECRET, UNCLASSIFIED, CONFIDENTIAL - CATEGORY A}. The ordering of these labels in terms of sensitivity is accomplished by a second set (P) defining a partial ordering on C_L. The levels in this set are called "compromise" levels to distinguish them from "integrity" labels that may be associated with objects and subjects in an integrity model (See Biba Integrity, Appendix C). If there are no other labeling schemes in the system besides compromise levels, the levels are often called "security levels."

The specification of the set of compromise levels as a level-one set (unconstrained and unordered) is a deviation from the original model. Security levels in BLP are defined as two-tuples with the first element coming from a totally ordered (hierarchical) set of clearance levels (e.g. {TS, S, C, U}) and the second element from the power set of an unordered (flat) set of categories. The partial ordering on the security levels is then defined in terms of the total ordering on the clearance levels and subsets of categories. For example, TS.A.B dominates S.A because TS is greater than S in the totally ordered set of clearance levels and {A} is a subset of {A,B}.

The levels specification in BLP has the advantage that the determination of dominance is a two-step operation and thus the algorithm is said to be "constant time" [5]. On the other hand, there are many problems with this scheme. The enumeration of these problems will be the subject of a future paper. The central point here is that we view the set of security levels is as a fundamentally a level-one set for maximum flexibility.

The final level-one set is that of access privileges. The specification of just a read-only (r) and blind-write (a) is a deviation from the original model. BLP specified four access rights: (1) read-only-observe but no modify, (2) execute - neither observe nor modify, (3) write - modify and observe, and (4) append - modify but no observe. Note that all of these rights are defined in terms of the two more primitive access privileges called "modify" and "observe." Therefore, in the abstract, these two primitives are the only ones required.

The level-two sets include the partial ordering (P) on the compromise levels (C_L), the mapping functions assigning classifications to objects and subjects (Fo and Fs), and the definition of the universe of all possible accesses between subjects and objects (M).

The set P is a set of two-tuples defining a partial ordering on the compromise levels (C_L). The relationship primitive "is-a-partial-ordering" is rigorously defined in Appendix A. It corresponds to the intuitive notion of dominance in terms of data sensitivity. Because of the transitive nature of the definition of partial ordering, all pairs that directly or indirectly dominate each other must be contained in the set. In other words, if (TS,S) and (S,C) are in P indicating that TS dominates S and that S dominates C, then (TS,C) must also necessarily be in the set. This may be impractical to implement for very large sets of compromise levels (C_L). For an implementation, the inclusion of only directly dominating pairs would suffice. Transitive closure of the set could be computed on the fly.

The sets Fo and Fs assign classification labels to objects and subjects, respectively. The method of making this assignment is by the use of the two-tuples where the first element is chosen from the set of objects (or subjects in the case of set Fs) and the second element is chosen from the compromise level set. For example, if (o1, TS) is a member of Fo, this means that object o1 is classified top secret. The sets Fo and Fs are equivalent to the similarly named mapping functions in BLP. In this representation, they will be referred to as the classification-mapping sets.

The classification-mapping sets depend on two different level-one sets. This is important since changes to any element in a level-one set on which a level-two set depends may adversely affect the level-two set as well. For example, a change made to the set of objects (e.g. when a subject writes to an object) will impact on Fo since it depends on the set of objects (O) and the set of classification levels (C_L). Furthermore, a change to C_L impacts on both Fo and Fs. This dependency is highlighted by the organization of SNF.

The definition of Fs is a deviation from the original model. BLP has two mapping functions for subjects: fs and fc. Both functions map subjects-to-levels. The level assigned to a subject by fs is the upper bound of the level a subject may take, whereas fc assigns the current level. This implies that fc is a dynamic assignment constrained by fs. This conceptual separation has been deleted from our representation for simplicity.

The separation of labeling functions described in the original BLP model is logically equivalent to having multiple subjects, each taking on one of the classifications allowed to the single subject under BLP. For example, subject s1 exists under the unmodified BLP scheme, it is currently classified "C," and the maximum level s1 can take is "S." Assuming the levels are ordered as {S,C,U}, this is logically equivalent to having three subjects, s11, s12, s13, classified at U, C, S, respectively. This shall be referred to as the one-subject-one-level approach.

The alternative one-subject-one-level approach is also more satisfying in that it promotes tranquility of the security state of the system. This is preferable in that changes made to any set by any operation on a system require a proof that the change to that set and any sets that depend on it do not corrupt the security of the system.

The final level-two set is M. It defines the universe of all possible accesses that each subject may have to each object. The security policy set, B, equal to the universe set, M, represents a completely permissive system in which each subject has all possible access privileges to each object in the system. The universe is then restricted by one or more security policies represented by subsets of the set M and intersected to form the overall policy.

As a point of clarification, the three-tuple (s1, o1, r) in set Bsp means that subject s1 has read access to object o1. It is equivalent to entering the access privilege r in the (1,1) cell of an access matrix. The use of set notation may seem awkward here at first, but it maintains consistency with the rest of the notation and allows a somewhat different perspective on access control than that provided by matrices.

The use of the letters M and B for these sets may be somewhat confusing since BLP's definition for the same terms are different. As defined in this paper, M is the cross product of the set of subjects (S), the set of objects (O), and the set of accesses (A) resulting in all possible combinations of elements chosen from each of these sets. BLP defines M as the access matrix indexed by subject and object. Each cell in the matrix M contains a set of allowable accesses. In this paper sets beginning with the capital letter B represent instantiations of security policies. The single letter B represents a security policy equal to the universe set M which implies no restriction to any access. In BLP, B is the power set of the universe set. In other words, this B represents all possible restrictions that may be placed on the universe. There appears to be no utility in using the power set of the universe in our context so we have chosen to leave it out.

The level-three sets represent a fairly significant departure from the original BLP model in form but not in content. BLP defines two rules that restrict the promiscuous universe of full access, each in its own way. The restriction of this set is implicit since it is specified by defining a state such that the accesses allowed adhere to the rules. SNF, on the other hand, makes the restrictions of the universe defined by a rule explicit by the specification of a set. This set ultimately determines the subset of the universe which implements the security policy associated with the rule.

The first level-three set, Bms, defines the mandatory security policy. Since mandatory security is really the combination of restrictions on read access by simple security and on write access by the *-property, the set Bms is broken up into two explicit subsets associated with each of the rules. These sets are unioned to form Bms-those accesses allowed under the integrated mandatory security policy.

The subset Bms_r represents the read restrictions imposed by the simple security property. The property has been summarized as meaning "no read up" in classification level. For example, a secret-cleared subject is prevented from reading data in a top secret object. Read access is permitted only if the level of the subject dominates the level of the object.

Similarly, the subset Bms_a represents the write restrictions by the *-property. This property has been summarized as meaning "no write down" in classification level. For example, a secret-cleared subject may not write to an unclassified object since secret information could flow to the object thereby exposing it to unclassified subjects. Write access is permitted only if the level of the object dominates the level of the subject.

The second level-three set, Bds, defines discretionary security policy. Bds is an arbitrary subset of the universe of promiscuous accesses defined in set B. The discretionary nature of the set comes from additional rules defining subsets of B over which each subject has dominion. Dominion means the discretion to include or exclude an element from Bds, thereby allowing or denying the corresponding access privilege.

The final level-three set, Bs, defines the unified security policy - the combination of mandatory and discretionary access control. The set Bms defines the mandatory security policy. The set Bds defines the discretionary security policy. These two set are intersected to form the unified policy. The fact that the combination is done by intersection says that if a given access privilege is denied by either policy, it is denied in the unified policy. In this way, even though an arbitrarily large subset of the promiscuous universe set, the discretionary access set cannot allow access denied by the mandatory security policy embodied in set Bms.

This representation of the total security policy as the intersection of subsidiary policies allows simple extension to include other security protection policies with these two by simply intersecting them to the final set. This, for example, makes addition of type enforcement described in Appendix C simpler to grasp in its relationship to the other security policies with which it coexists.

Biba Integrity Model

The Biba Integrity Model is essentially an exact parallel to the BLP model with compromise levels (C_L) replaced by integrity levels (I_L). The SNF description in Appendix B is nearly identical to that of BLP with the above noted change propagated throughout.

The model is intended to protect the integrity of data in a system so as to prevent unauthorized modification of objects. For example, a data base locating the space junk orbiting the earth may be unclassified, but the integrity of this information is critical to plotting a safe course for space craft. The Biba Integrity Model attempts to deal with this problem by adding integrity-related labels to all of the subjects and objects and restricting access to protect critical files (objects).

The innovation in the model is not in its form (since it is equivalent to BLP) but rather in the assignment and interpretation of labels. It is difficult to make sense of assigning integrity labels to subjects. The assignments necessary to provide integrity protection of certain types seems contorted at times. Indeed, the author drops parts of the parallel to BLP due to an inability to ascribe a meaning to them.

The attempt to parallel BLP resulted in a somewhat contorted model that was not powerful enough to fulfill the spectrum of requirements of integrity enforcement. For example, the model cannot protect intermediate file results in a pipeline of programs without requiring a substantial amount of trusted software.[3] One of the most important goals of a model is to minimize the amount of trusted software since software verification is expensive. This leads the discussion to the next section on role enforcement. This model addresses the same integrity problem, but with much more power.

Role Enforcement Model

The level-one sets of objects (O), subjects (S), and accesses (A) are as in the BLP model description above. The set of types (T) and domains (D) are new sets that will act as an orthogonal label set for objects and subjects respectively. The unique aspects of role enforcement [3] are based on these two sets.

The first level-two set, F1, assigns types to objects. Notice that the mapping need only be complete - each object must have at least one type associated with it. There appears to be no need to make the mapping unique as for classification labels in the standard BLP model. In other words, there appears to be no needed restriction in this policy that prohibits an object from being of more than one type. Similarly, the set F2 maps all subjects to domains.

The universe of all possible accesses is again defined as M as in Appendix B for BLP.

The level-two set, F3, defines the universe of all possible accesses between domains and types just as is done above for the set M. Indeed, if each object is assigned a unique type, and each subject is assigned a unique domain, F3 is isomorphic to the universe set M.

The first level-three set, F4, is defined as a subset of F3 that defines a particular access policy. F4 is an arbitrary subset of F3 in very much the same sense that Bds is an arbitrary subset of B. Indeed, under the special condition stated above, the analogy is exact. This brings up an interesting point. Is this model a kind of discretionary access control, and does it suffer from the same inherent weaknesses? The answer is no, but only if the assignment of labels is done carefully and the mappings of objects-to-types and subjects-to-domains remains tranquil (static). Restricted changes could be allowed, but they would have to adhere to some stated properties if they are not to corrupt the integrity protection.

The set Brs is a level-three set that essentially maps the access restrictions imposed between domains and types back to restrictions between subjects and objects. Brs, therefore, defines a particular role enforcement security policy. It is a defined subset of the universe of all possible accesses (M) in the same way that Bds (discretionary policy) and Bms (mandatory) are also subsets of M. Role enforcement is represented as just another subsidiary policy in the same form that can simply be intersected into the unified policy defined by the set Bs1.

The unified policy set, Bs1, is defined as the intersection of the BLP unified policy Bs (which combines mandatory and discretionary security - see Appendix B) and the policy enforced by role security. This demonstrates the facility of adding coexisting security policies under SNF.

MLO Model

In 1985 SYTEK, Inc., produced an MLO Model under a Rome Air Development Center contract. At that time, we were not able to compare the MLO model rigorously with any other model due to the lack of a standard notation. In casting the MLO model in SNF we were able to grasp its content. In this paper we cast that part of the MLO model which allows us to compare it with BLP. A complete casting of the MLO model will appear in a subsequent paper.

MLO access attributes are treated at a higher level than BLP. In addition to the BLP read and write, the actual MLO set of access types includes the following: create, destroy, downgrade, upgrade, owner, clearance, Discretionary Access Table (DAT), and kill. For simplifying the comparison with BLP, we will consider only the read and write access attributes.

The set of security levels (S_L) is the equivalent to the BLP SNF set of compromise levels (C_L). The subject and object sets are BLP equivalents also. The set of roles (RO) list all the possible roles under which subjects may operate. Subjects may operate in one role at a time (S_RO).

The level-two sets, partial ordering (P) and object-level mapping (Fo), are BLP SNF equivalents. In MLO, however, we have two security levels associated with each subject: the container clearance (Fc) for reference path access and the data clearance (Fd) for object access.

To model the relationship between multilevel objects (i.e. containers) and single level objects (atoms) the set H is defined with the primitive is-a-hierarchy-on (see Appendix A). The set H models both the container-atom relationship between objects as well as determines the possible reference paths for object access. For example, if an object oz is contained in container oy which is, in turn, contained by container ox, a reference path to oz would be the ordered tuple <ox,oy,oz>. The set Q represents all such sequences in H. H also determines which objects are atoms and which are containers by using the convention that the leaves are the atoms.

The reference mechanism in the MLO model returns the reference path which must be used to access an object. The set RF models this mechanism and maps subject-role pairs and objects to reference paths. There is only one allowable path to an object for each subject and role combination. Frp, a level-five set, models the association of a security level with each path. Given a subject in a specific role accessing an object, there is only one path allowed, and it has a single security level associated with it. For example, in a top secret document there is an unclassified paragraph which may be only accessed if the user has a top secret clearance. The reference path here is opening the document and then reading the paragraph. This is modeled by the use of a container clearance per subject and assigning top secret to all reference paths to the paragraph. The subject's container clearance must dominate the level of the reference path used to access the object.

Finally, we create the sets Bms_r and Bms_w which are analogous to the BLP Bms_r and Bms_a sets. An element (sr,o,r) of Bms_r implies that read access is allowed because the subject (s), acting in the role (ro), has a container clearance which dominates the security level of the reference path to object (o) and a data clearance which dominates the object's security level. This is the MLO equivalent to simple security.

The MLO equivalent to the *-property is the set Bms_w. An element (sr,o,w) of Bms_w implies that write access is allowed because the subject (s), acting in the role (ro), has a container clearance which dominates the security level of the reference path to the object (o) and a data clearance which is <u>dominated by</u> the security level of the object.

The unified security policy is determined as in BLP. Given an arbitrary set, Bds, which represents the discretionary access controls and the set Bms which is the union of Bms_r and Bms_w, the total secure access set is the intersection of Bms and Bds.

In simplifying the MLO model to compare it with BLP we ignored the concept of users and operations. The MLO model is much more comprehensive than we present here; however, we feel we have accomplished our purpose of comparing it with BLP. The interesting problem of modeling parameters in set theory will be solved in the full casting of the MLO model in SNF.

CONCLUSION

The notation proposed, SNF, is consistently based on set theory representations. This has proven sufficiently powerful to represent the essence of four different security models.[1]

Merely representing these models in SNF has given the authors new insights into the meaning and ramifications of these models. SNF promises to greatly facilitate the analysis of existing models and the comparison between models. Several follow-up papers based on SNF are planned.

[1] We expect SNF to be rich enough to represent the full subtlety of current computer security models, however this remains to be shown by future analysis.

REFERENCES

1. Bell, D.E. and L. J. LaPadula, "Secure Computer System: Unified Exposition and Multics Interpretation," (Bedford, MA: Electronic Systems Division, AFSC, Hanscom AF Base), January 1976, ESD-TR-75-306.

2. Biba, K., "Integrity Considerations for Secure Computer Systems," (Bedford MA, Electronic Systems Division, AFSC, Hanscom AF Base), April 1977, ESD-TR-76-372.

3. Boebert, W.E. and R. Y. Kain, "A Practical Alternative to Hierarchical Integrity Policies," Proceedings of the 8th National Computer Security Conference, (Gaithersburg MD: DOD Computer Security Center/National Bureau of Standards), September 1985, pp 18-27.

4. Sullivan, E.C., T. F. Lunt, and N. Proctor, "The Multilevel Object Security Model," (GRIFFISS AFB, New York: Rome Air Development Center), November 1985, F30602-85-C-0001.

5. Aho, A., J. Hopcroft, and J. Ullman, The Design and Analysis of Computer Algorithms, Addison-Wesley, 1974, Chapter 1.

APPENDIX A: Notation Primitives

Set Normal Form Construct Definitions
07/18/86

Strings of ASCII characters are used in place of standard mathematical symbols for two reasons: we wish to easily transmit this document electronically, and we do not have the graphics capability to support editing such a document. Realizing that these symbols are near and dear to the mathematician's heart, we will produce this notation, sometime in the future, using standard mathematical symbols.

1. maps-completely-to: Given sets A and B and M := { (a,b) | a member-of A and b member-of B}, A "maps-completely-to" B iff for_every a member-of A there_exists (a,b) member-of M for some b member-of B.

2. maps-uniquely-to: Given sets A and B and M := { (a,b) | a member-of A and b member-of B}, A "maps-uniquely-to" B iff (a,b1) member-of M and (a,b2) member-of M implies b1=b2 where a member-of A and b1,b2 member-of B.

3. maps-fully-to: Given sets A and B, A maps-fully-to B iff A maps-uniquely-to B and A maps_completely to B

4. PS(A): power set of A

5. is-a-partial-ordering-on: Given set A and P := { (a1,a2) | a1, a2 member-of A }, P "is-a-partial-ordering-on" A iff

(i) a1 = a2 or
(ii) (a1,a2), (a2,a1) members-of P => a1=a2 or
(iii) (a1,a3), (a3,a2) members-of P => (a1,a2) member-of P where a1,a2,a3 members-of A

Comment: The three conditions specify reflexivity, antisymettry, and transitivity required by a partial ordering. P captures the dominance relationship described by BLP.

6. is-a-hierarchy-on: Given set A and H := { (a1,a2) | a1,a2 member-of A } H "is-a-hierarchy-on" A iff

(i) a1 not = a2 and
(ii) for all sequences of members of H where (a1,a2),(a2,a3),...,(ai,ai+1),...,(an-1,an) is a sequence where the second element of one pair is first element of the next pair => a1 not = an

Comment: The two requirements specify that (1) no container contains itself, and (2) there are no cycles within the representative digraphs.

7. is-a-leaf-in: Given H is-a-hierarchy-on A then a is-a-leaf-in H iff

(i) for all (ai,aj) member-of H, a not = ai

Comment: There are no objects which this object contains. By our restriction that all containers must contain at least the NULL atom, only the atoms will satisfy the above criteria.

8. is_a_subset_of: A is_a_subset_of B iff

(i) for all a member-of A, then a member-of B

APPENDIX B: Bell And Lapadula

Modified Bell and Lapadula Model
Set Theory Casting
08/29/85, 09/18/85

Level-one Sets: The fundamental sets of the modeling system.

C_L := set of all compromise levels
O := set of all objects (data;files;pgms;subjects;i/o devices)
S := set of all subjects (processes;pgms in execution)
A := (r,a) the set of access rights
r means read-only access
a means blind-write access

Level-two Sets: The sets which depend only on fundamental sets.

P is-a-partial-ordering-on C_L

Def: l1 R l2 denotes the dominance relation R between C_L members l1 and l2. l1 R l2 iff (l1,l2) is a member of P. In BLP terms, l1 R l2 means level l1 dominates level l2.

Fo := { (o,l) | o member-of O, l member-of C_L, and O maps-fully-to C_L}

Comment: Fo is a set that equivalently specifies the BLP function Fo: O --> C_L.

Def: Given a particular (o,l) member-of Fo, Fo(o) refers to the level l to which object o is mapped in the tuple.

Fs := { (s,l) | s member-of S, l member-of C_L and S maps-fully-to C_L}

Comment: Fs is a set that equivalently specifies the BLP function Fs: S --> C_L.

Def: Given a particular (s,l) member of Fs, Fs(s) refers to the level l to which subject s is mapped in the tuple.

M := { (s,o,x) | s member-of S, o member-of O, x member-of A}

Comment: M is the set of all possible access between subjects and objects independent of the mapping functions. M is essentially equivalent to an access matrix with all of the entries filled in with full access.

Level-three Sets: The sets which depend on level-two sets.

Bms := Bms_r Union Bms_a

Comment: Bms is a subset of M which defines the access between subjects and objects that are allowed by simple-security and the *-property.

Bms_r := { (s,o,r) | Fs(s) R Fo(o) }

Comment: Bms_r are those accesses allowed by simple-security.

Bms_a := { (s,o,a) | Fo(o) R Fs(s) }

Comment: Bms_a are those accesses allowed by the *-property.

Bds is_a_subset_of M

Comment: Bds are those accesses that are allowed by discretionary security. BLP refers to this as matrix M, where M(i,j) = r means that the ith subject has read access to the jth object. This is represented by the triple (si, oj, r) in the set Bds in SNF.

Bs := Bms Intersect Bds

Comment: Bs corresponds to the set of all secure access triples. This set defines all accesses allowed in a given security system.

APPENDIX C: Biba Integrity

Integrity Parallel of the Modified Bell and Lapadula Model
Set Theory Casting
09/03/85, 9/18/85

Level-one Sets: The fundamental sets of the modeling system.

I_L := set of all integrity levels
O := set of all objects (data;files;pgms;subjects;i/o devices)
S := set of all subjects (processes;pgms in execution)
A := {r,a} the set of access rights
r means read-only access
a means blind-write access

Level-two Sets: The sets which depend only on fundamental sets.

P is-a-partial-ordering_on I_L

Def: l1 R l2 denotes the dominance relation R between I_L members l1 and l2. l1 R l2 iff (l1,l2) is a member of P. In BLP terms, l1 R l2 means level l1 dominates level l2.

Fo := { (o,l) | o member-of O, l member-of I_L and o maps-fully-to l}

Comment: Fo is a set that equivalently specifies the BLP function Fo: O --> I_L.

Def: Given a particular (o,l) member of Fo, Fo(o) refers to the level l to which object o is mapped in the tuple.

Fs := { (s,l) | s member-of S, l member-of I_L and s maps-fully-to l}

Comment: Fs is a set that equivalently specifies the BLP function Fs: S --> I_L.

Def: Given a particular (s,l) member of Fs, Fs(s) refers to the level l to which subject s is mapped in the tuple.

M := { (s,o,x) | s member-of S, o member-of O, x member-of A}

Comment: M is the set of all possible access between subjects and objects independent of the mapping functions. M is essentially equivalent to an access matrix with all of the entries filled in with full access.

Level-three Sets: The sets which depend on level-two sets.

Bms := Bms_r Union Bms_a

Comment: Bms is a subset of M which defines the accesses between subjects and objects that are allowed by integrity simple-security and the integrity *-property.

Bms_r := { (s,o,r) | Fs(s) R Fo(o) }

Comment: Bms_r are those accesses allowed by integrity simple-security.

Bms_a := { (s,o,a) | Fo(o) R Fs(s) }

Comment: Bms_a are those accesses allowed by the integrity *-property.

Bds is_a_subset_of M

Comment: Bds are those accesses that are allowed by discretionary security. BLP refers to this as matrix M, where M(i,j) = r means that the ith subject has read access to the jth object. This is represented by the triple (si, oj, r) in the set Bds in SNF.

Bs := Bms Intersect Bds

Comment: Bs corresponds to the set of secure access triples. This set defines all access allowed in a given security system.

APPENDIX D: Role Enforcement

Type-Domain Mechanism Model Extension to the Modified Bell and LaPadula Model
Set Theory Casting
09/03/85

Level-one Sets: The fundamental sets of the modeling system.

O := set of all objects
S := set of all subjects
A := {r,a} the set of access rights

T := set of all types
D := set of all domains

Level-two Sets: The sets which depend only on fundamental sets.

F1 := { (o,t) | o member-of O, t member-of T and o maps-completely-to t}

Comment: F1 maps every object o to some type t. F1 corresponds to a mapping function F1: O --> T.

F2 := { (s,d) | s member-of S, d member-of D and s maps-completely-to d}

Comment: F2 maps every subject s to some domain d. F2 corresponds to a mapping function F2: S --> D.

M := { (s,o,x) | s member-of S, o member-of O, x member-of A}

Comment: M is the set of all possible access between subjects and objects independent of the mapping functions. M is essentially equivalent to an access matrix with all of the entries filled in with full access.

F3 := { (d,t,x) | d member-of D, t member-of D, x member-of A}

Comment: F3 is the set of all possible accesses between domains and types.

Level-three Sets: The sets which depend on level-two sets.

F4 is_a_subset_of F3

Comment: F4 represents an access matrix between domains and types.

Brs := { (s,o,x) | s member-of S, o member-of O, x member-of A and (d,t,x) member-of F4 and (s,d) member-of F1 and (o,t) member-of F2}

Comment: Brs is the set of all accesses allowed between subjects and objects in the type-domain model.

Bsl := Bs Intersect Brs where Bs is the BLP secure set

Comment: Bsl represents the logical AND of the basic access rights defined by the BLP model and the type-domain extension to that model.

APPENDIX E: MLO Model

Modified MultiLevel Object Model
Set Theory Casting
05/26/86

Level-one Sets: The fundamental sets of the modeling system.

S_L := set of all security levels
O := set of all objects (data;files;pgms;subjects;i/o devices)
S := set of all subjects (processes;pgms in execution)
A := {r,w} the set of access rights
r means read access
w means write access
RO := set of all roles

Level-two Sets: The sets which depend only on fundamental sets

P is-a-partial-ordering_on S_L

Def: l1 R l2 denotes the dominance relation R between S_L members l1 and l2. l1 R l2 iff (l1,l2) is a member-of P. In BLP terms, l1 R l2 means level l1 dominates level l2.

Fo := { (o,l) | o member-of O, l member-of S_L and O maps-fully-to S_L}

Comment: Fo is a set that equivalently specifies the MLO function Fo: O --> S_L.

Def: Given a particular (o,l) member-of Fo, Fo(o) refers to the level l to which object o is mapped in the tuple.

Fc := { (s,l) | s member-of S, l member-of S_L and S maps-fully-to S_L}

Comment: Fs is a set that equivalently specifies the MLO container clearance.

Def: Given a particular (s,l) member of Fc, Fc(s) refers to the level l to which subject s is mapped in the tuple.

Fd := { (s,l) | s member-of S, l member-of S_L and S maps-fully-to C_L}

Comment: Fd is a set that equivalently specifies the MLO data clearance.

Def: Given a particular (s,l) member of Fd, Fd(s) refers to the level l to which subject s is mapped in the tuple.

H := { (o1,o2) | H is-a-hierarchy-on O & (Fo(o1),Fo(o2)) member-of P}

Comment: This hierarchy determines the container-content relationship. Containers contain references to other containers and atoms. Atoms may be leafs and can only contain data. In order to maintain the leaf (atom) - non-leaf (container) distinction, we have adapted the convention that all empty containers contain a null atom.

SR := { (s,ro) | s member-of S, ro member-of RO}

Comment: SR is set of all permissable subject-role combinations.

S_RO is_a_subset_of SR and S maps-fully-to RO in S_RO

Comment: Every subject must be associated with only one role at any given time.

Level-three Sets: The sets which depend on level-two sets.

M = { (sr,o,x) | sr member-of SR, o member-of O, x member-of A}

Comment: M is the set of all possible access between subjects and objects independent of the mapping functions. M is essentially equivalent to an access matrix with all of the entries filled in with full access.

Q := { q | q is an ordered tuple <o1,...,on> & for $1 \leq i < n$, $n \geq 2$, oi member-of q, then (oi,oi+1) member-of H }

Comment: Set of all possible paths constructed of pairs of objects from H

AT := { o | o member-of O, o is-a-leaf-in H}

Comment: In the MLO model, atoms can contain only data at a single security level.

C := { o | o member-of O, not o is-a-leaf-in H}

Comment: In the MLO model, containers can contain only descriptors of other containers or atoms.

Level-four Sets: The sets which depend on level-three sets.

RF := { (sr,o,q) | sr member-of SR, o member-of O, q member-of RF, (SR,O) maps-fully-to RF, o=on where on is last oi in q}

Comment: This equates to the MLO reference mechanism. Given a subject-role pair and an object combination, there is exactly one reference path allowed.

Level-five Sets: The sets which depend on level-four sets.

Frp := { (sr,o,l) | (sr,o,q) member-of RF, l member-of L}

Comment: Frp(o) will be used as a shorthand to indicate the reference path level associated with an object for a given subject-role pair.

Level-six Sets: The sets which depend on level-five sets.

Bms := Bms_r Union Bms_w

Comment: Bms is a subset of M which defines the access between subjects and objects that are allowed by MLO equivalents to the BLP simple-security and the *-properties.

Bms_r := { (s,o,r) | Fc(s) R Frp(o) and Fd(s) R Fo(o) }

Comment: Bms_r are those accesses allowed given that the subject's container clearance dominates the reference-path level for the object and the subject's data clearance dominates the object's security level. (NO READ UP)

Bms_w := { (s,o,r) | Fc(s) R Frp(o) and Fd(s) R Fd(s) }

Comment: Bms_w are those accesses allowed given that the subject's container clearance dominates the reference path level for the object and the subject's data clearance <u>is dominated by</u> the object's security level. (NO WRITE DOWN)

Bds is_a_subset_of M

Comment: Bds are those accesses that are allowed by discretionary security. MLO defers this to implementation detail. In essence it is an arbitrary subset of M.

Bs := Bms Intersect Bds

Comment: Bs is only part of the full MLO model created by SYTEK, Inc. We put as much of the MLO model in SNF as necessary to compare it to BLP. We plan to put the full MLO model in SNF as a future paper.

RESEARCH TOWARD INTRUSION DETECTION THROUGH AUTOMATED ABSTRACTION OF AUDIT DATA

JEFFREY D. KUHN
National Computer Security Center
911 Elkridge Landing Road
Linthicum, Maryland 21090

ABSTRACT

Auditing seldom plays a role in detecting illegal attempts to access data residing in computers. Instead, if audit data are used at all, it is generally in the form of detailed printouts that are pored over by the security officer for further evidence of wrongdoing after illegal activity has already been discovered. This paper examines the issues involved in using audit data to detect illegal computer activity, and proposes an audit system based upon the results of that examination.

INTRODUCTION

There are two major sources of audit data typically produced by timesharing operating systems to provide a history of system use. An accounting system provides the information necessary to bill account holders for the computer time that they use, and security logs provide listings of attempts to use priviledged commands. The information collected this way that indicates a security violation, if it exists at all, is usually too well dispersed within a large volume of similar but irrelevant data to be useful for the detection of that violation. Instead, it is generally used only to confirm something already strongly suspected, or to add additional evidence to that already in existence.

Despite the poor performance of present auditing techniques applied to security, a combination of proper audit data and tools for the computer aided analysis of that data should provide a security officer with the ability to identify some illicit computer activities. The remainder of this paper will examine activities that violate the security of computer systems, determine what audit information needs to be collected, and describe how that information can be analyzed to detect undesirable activity.

VIOLATING SECURITY

At the most fundamental level, the methodology for compromising information security is the same whether that information is contained within a computer system or not. A compromise occurs through some combination of a violation of trust and a circumvention of physical and procedural safeguards. If the violation involves aspects of security not intimately related to the use of a computer, then there will probably be little to distinguish the computer activity involved. In this situation the computer is merely a tool being used in the proper manner.

The situation of interest is one where the safeguards that are being abused exist within the operating system of a computer. If this is the case there may be little to distinguish the illicit act except for information regarding particular computer activity. The computer is then no longer a properly used tool but is instead a fundamental part of the compromise. This type of violation or circumvention of operating system safeguards is commonly referred to as system penetration.

A PHILOSOPHY OF PENETRATION

When a person performs what appears to be a single operation on a computer, they are interacting with the computer operating system at a level of abstraction above what is actually occuring. The operating system that the user manipulates is an abstract model that has been implemented with lower level operations. A typical operating system contains several such levels, each model implemented in the levels below it. The lowest level operations are implemented in hardware. It is an interpretation of the effects of several hardware operations that the user recognizes as the result of any individual command.

A properly constructed implementation must exhibit all the properties that will allow it to be recognized as the correct abstraction. Conversely, it should not display any properties not shared with abstraction being implemented. Doing this perfectly is a very difficult thing to ensure. The typical operating system has a number of implementation flaws at each level of abstract operation. When such an inconsistency is discovered, it can often be employed by a sophisticated user to perform operations that would otherwise be disallowed as violations of security.

Discovering and exploiting inconsistent implementations is only one technique employed in the compromise of computer security. At its most abstract level, an operating system is still complicated enough that administrative oversights and design errors will sometimes exist that can lead to properly unauthorized access to sensitive data. In such cases, penetration occurs despite a potentially correct implementation of the operating system design because security has not been correctly considered either by the system designers or the system administrators. Whether the error is made by the implementor, the designer, or the administrator, penetrations are effected by achieving an understanding of the operating system and considering the ramifications of using a command or set of commands in an unanticipated manner.

APPLICATION OF AUDITING TO PENETRATION

An auditor must collect data at a relatively low level of implementation for it to be effective in the discovery of penetrations. This ensures that most of the flaws that can be exploited exist at levels of abstraction higher than that being audited, maintaining the integrity of the audit data. A second requirement necessary to maintain the integrity of the data is for the auditor and the collected data to be housed in a processor distinct from that being audited. Otherwise, the successful penetrator might be able to erase or alter the data. Collecting the data at such a low level of implementation only increases the problems associated with the large volume of audit data involved, leading to a third requirement that a large percentage of the potential data not be generated at all or else be disposed of early in the auditing process. The remaining data should then be processed by summarizing the effects of the low level commands upon data objects representing relatively high level concepts of system security. Whenever possible, these data objects should become the raw data that is examined and further processed for evidence of security violations so that the volume of audit data that must be searched to find a penetration attempt is reduced.

The security officer with this kind of an auditor will be able to monitor system security by noting the access of special purpose files used by the operating system, and by watching the use of commands with particular relevance to system integrity. Known flaws can be carefully watched. Trojan horse programs and viruses can, in some cases, be identified by their access of files and patterns of information flow. Some covert channel manipulations can be detected by their distinctive use of system calls. The general technique is to characterize specific penetration techniques and identify their use in system activity. An important point is that much of the information used in the characterizations will be system specific.

The intent is to put reliable information into the hands of the security officer that has a direct bearing on possible attempts to circumvent the security controls built into the operating system and achieve unauthorized access to data. The information is retained on-line so that it can be further processed in a manner directed by the security officer. This approach uses the unique capabilities of both computer and human to positive advantage: the computer's ability to quickly organize and process data and the human talent for recognizing relevant situations and interrelationships.

THE APPROACH

The above approach was applied to auditing the UNIX operating system. UNIX was chosen for several reasons. First among them was expediency; I have a PDP 11/70 running a version of UNIX and several UNIX experts at my disposal. Second, the UNIX system is an almost ideal candidate. It was developed as a simple but powerful, general-purpose operating system. The fundamental generality of the individual commands makes it possible to use them in often completely inappropriate ways, tremendously increasing the possibilities of finding an unanticipated combination of commands and arguments leading to a violation of computer security. Modifying UNIX to produce the audit data needed is easy because it is written and maintained in C, a high-level language.

Like most multiuser computers, the PDP-11/70 simulates multiprocessing on a computer with only a single processor. The hardware is designed such that virtual memory is mapped to actual memory in a way enabling the address space of an individual process to be kept distinct from that of other processes. A process must access all resources other than its own distinct address space through requests to the operating system, which mediates the requests and enforces the concept of process separation.

The UNIX operating system is designed around the central concept of a file system. All resources are represented as files, greatly simplifying their access. Process separation is maintained by requiring that processes have the proper permission before they can access a file. Each file is marked with a permission field that indicates what class of processes may access it and how it may be accessed. The permission field indicates read, write, and execute permissions for three types of processes: processes owned only by the owner of the file, processes owned my a member of the owner's group, and by any process regardless of ownership. There is an owner named root with special privilege. root has ownership of the files representing the disks, core, and so forth. Processes owned by root also have the privilege of accessing any file regardless of the file's permission field.

UNIX is implemented in the C programming language, augmented with operations called syscalls. Syscalls enforce the abstractions of a file system and the necessity for processes to have the privilege to access files. This is the level at which I chose to record audit data. It requires that the syscalls properly employ the hardware based memory mapping to maintain the illusion of virtual memory, and that the file system abstraction is both correctly implemented and properly enforces security. There is also some circularity involved, since the C language and the syscalls run on the UNIX operating system. While these assumptions are not necessarily completely valid, enough flaws exist beyond any in the syscalls and the file system to make it worthwhile.

It was decided to use syscall audit data to construct representations of the propagation of priviledge (indicating what each process may access), process lineage (keeping track of process ownership), and file system manipulation by each process (identifying potential information flow). These representations would be the data submitted to the security officer for further analysis. Making the above assumptions, this information should be sufficient to identify many potential breaches in information security that can occur through exploitation of the UNIX operating system.

About half of the fifty-six syscalls implemented on our version of UNIX (a modified version 6) were identified as having an impact on the state of the abstractions chosen for display to the security officer. After careful consideration, twenty of these were chosen for auditing. The goal was to produce the most accurate representations possible with the least amount of auditing and the smallest amount of processing. Syscalls such as pause were rejected as having no direct impact. Others, such as read were rejected because of their high frequency of use, and because their use can be assumed from the use of the open syscall.

A Symbolics LISP machine was chosen to receive and process the raw audit data. The Symbolics machine was chosen because of its suitability for symbolic manipulation of lists, capability for prototyping, and attention to tools for the construction of elegant user interfaces. A major goal is the creation of a system which is easy to use. The security officer will be able to run background processes that screen the incoming data for particular events, and perform further analysis in a batch mode.

CONCLUSION

An examination of operating system penetration techniques and current auditing methods indicates that most sophisticated violations of system security will be completely undetected, leaving potentially no trace in the audit logs at all. Perhaps the only method of identifying this type of violation is to characterize specific classes of penetrations and attempt to recognize their occurrance. To do this, however, it is necessary to audit data at a lower level of system implementation than is currently the practice. This data must then be processed, both automatically and with human guidance, to identify individual penetration attempts. In order to be effective, the auditor must, as much as possible, represent the audit data and penetration characterizations in terms of high level abstractions rather than the low level audit data. This reduces the amount of further processing that must take place. It is also important to give the security officer as much power as possible to describe characterizations of security violations and guide the resulting search for their occurrance. Above all, the proposed system is a tool involving human participation.

BIBLIOGRAPHY

Peters B., "Computer Security Today," *Proc. 7th DOD/NBS Computer Security Conference*, pp. 270-276, 1984.

Ritchie D., and K. Thompson, "On The Security Of Unix," *Documents For Use With The UNIX Timesharing System*, 6th ed., Bell Laboratories, Murray Hill, NJ 07974.

Ritchie D., and K. Thompson, "The UNIX Timesharing System," *Documents For Use With The UNIX Timesharing System*, 6th ed., Bell Laboratories, Murray Hill, NJ 07974.

Thompson K., "Reflections on Trusting Trust," *Communications of the ACM* vol. 27, no. 8, August 1984.

Wood P. H., and S. G. Kochan, *UNIX System Security*, Pipeline Associates Inc., Hayden Publishing Company Inc., Hasbrouck Heights, NJ. / Berkeley California, 1985.

TRUST ISSUES OF MACH-1

Dr. Martha A. Branstad, Ms. Pamela S. Cochrane,
Dr. D. Elliott Bell, and Mr. Stephen T. Walker
Trusted Information Systems, Inc.
P.O. Box 45
Glenwood, MD 21738

I. INTRODUCTION

Trusted Information Systems, Inc., is investigating the feasibility of creating a trusted version of the Mach-1 operating system being developed at Carnegie-Mellon University. Initial analysis is being done on Accent, the progenitor of Mach-1, since both systems are message-based and focus on ports (kernel managed message queues), as the central abstraction, although Accent has a simpler process structure and no memory sharing. Accent is a well-structured system designed with protection as a system goal. Consequently, the crucial trust issue in determining if Accent can be made to conform with the DoD Trusted Computer Security Evaluation Criteria (TCSEC) for a level B3 system is the ability to associate labels with subjects and objects within the system to support mandatory access control.

Two different approaches to labeling, each at a different level of abstraction with respect to the system, merit further investigation. These approaches are: 1) associate labels with ports and processes managed by the kernel; and 2) modify the existing access group structure and use the Authentication, Authorization, and Name Servers to provide mandatory access control. This paper will examine the structure of Mach-1 and Accent, constraints imposed by the TCSEC, and the two approaches to labeling outlined above. This is a preliminary report on a research project in its early phases; it presents initial findings and strategies, not completed research.

II. MACH-1 AND ACCENT

Mach-1 is the kernel of a distributed operating system designed for a diverse set of machines, ranging from workstations to very high performance multiprocessors. Mach-1 is based upon the Accent kernel used in the Spice distributed operating system at Carnegie-Mellon University.

Since Mach-1 bears a strong conceptual similarity to Accent, we will first discuss Accent, the simpler of the systems. Accent is designed as a message-based system, with processes communicating via messages. Inter-process communication, process management, and virtual memory management are handled by the Accent kernel. Other operating system services are provided by servers outside of the kernel.

Messages in the system are sent and received via ports which are kernel-managed and protected queues for messages. Access rights for ports have three types: Own, Receive, and Send. Own and Receive are unique rights; only one process may possess Receive (Own) rights for a given port at any one time. Many processes, however, may have Send rights to a port at the same time. Access rights may be passed in messages. Each process has (capabilities for) a pair of ports used for communication with the kernel. The kernel maintains records of the port capabilities associated with each process, providing control of port creation and propagation. Each process has a large virtual address space; Accent does not support memory sharing. Through commands to the kernel, process creation (deletion), message sending (receiving), and memory access are achieved.

Mach-1 is being designed to run efficiently on multiprocessors. The differences between Mach-1 and Accent reflect this difference in design goals. Mach-1 maintains the message-based paradigm of Accent, with ports and port access rights remaining the same. Processes, however, are represented with separate abstractions for the environment and the executable portion of the process. A task is the environment in which a collection of "lightweight" processes termed "threads" may execute. Ports are associated with the task, although specific ports may be designated as primarily associated with a specific thread. Protection is associated with the task, since threads all share the environment provided by the task. To facilitate multiprocessor operations, memory sharing is supported by Mach-1.

In Mach-1, as in Accent, the kernel mediates communication via messages. The kernel stores the access rights for ports and determines if message transmissions are permitted. Messages which carry access rights in their contents must be so designated, and the kernel updates its tables as appropriate when such rights are transferred. Unless the kernel has information concerning the port access rights, the access designators are not effective.

Perhaps in contrast to what might be expected in the development of a distributed system by a university research group, protection has been a design goal for both Accent and Mach-1. This view of protection includes process isolation, user authentication, authorization for use of services, and discretionary access control. It does not include a parallel to military security labeling (classification and clearances) and mandatory access control. A structure to support protection is an integral part of the basic system design.

Kernel mediation of message communication via control of port access rights is a central mechanism of both operating systems. The distributed system is organized so that major server functions exist at both local and central sites. Global data stores and system records are kept at central sites which are physically secured. Local sites provide local functionality and communicate with central servers through protocols that can authenticate servers to one another, and to users. Encryption may be used to secure communication lines. Special servers handle user identification and authentication on the system. As indicated, protocols of communication with central authentication servers can be used to authenticate users to servers, and vice versa. Central servers can act as key distribution centers.

Discretionary access control is provided by the Authorization Server and the Name Server in conjunction with the Authentication Server. The Authorization Server maintains access group membership for each user via values for two entities, the primary and secondary access groups. Group membership is determined at login and communicated to the Authentication Server. The Name Server maintains Access Control Lists (ACLs) associated with each named object. When files or services are requested from the Name Server, it compares the access group membership (acquired from the Authentication Server) against the ACL to determine if access to the named object is authorized.

Although there has been considerable concern for and attention paid to providing protection in Mach-1 and Accent, neither would currently qualify as a DoD trusted computer system with more than discretionary access control. Missing from both systems is any mechanism that corresponds to sensitivity classes for subjects and objects. Sensitivity labels and a mechanism that uses them to enforce mandatory access control must be added to the CMU systems to provide a basis for a stronger trusted computing capability. For our investigations we have targeted B3 as the goal.

III. APPROACHES TO MANDATORY ACCESS CONTROL IN ACCENT

Discussions with CMU researchers have probed two different approaches to providing mandatory access control: 1) associate labels with ports and processes managed by the kernel, and 2) modify the existing access group structure and use Authentication, Authorization, and Name Servers to provide mandatory access control.

Approach 1, Kernel Mediation, would require an extension to the port access mechanism to provide sensitivity labeling. The kernel would check for sensitivity label mis-match while mediating port access rights. Although the Kernel Mediation approach would involve modification to the kernel, the basic mediation mechanism already exists. This approach is consistent with the TCSEC B3 level mechanism for mandatory access control with minimization of the TCB.

Approach 2, Server Mediation would use existing Accent servers to provide mandatory access control as well as the discretionary control they currently provide. Modifications to both data structures and control code in the servers would be required, but no modification to the kernel itself is anticipated. This approach, which labels and controls objects visible to users of the system, should be adequate to support a B2 or B3 system.

Both of the above approaches will be discussed in greater detail in the remainder of this paper, although the Kernel Mediation approach has been the focus of most of our energy. It should be noted that either approach, when transferred to the Mach-1 system, must deal with the issue of multiple threads (executable units) in a single protection domain provided by a task; an issue in conflict with literal interpretation of TCSEC requirements for an isolated protection domain for each process. Since threads appear to be fundamental to achieving effective performance for multiprocessor systems, however, this separable issue should be considered as a point for future interpretation of the TCSEC.

A number of other features must be provided in order to qualify as a trusted computing base (e.g., auditing, trusted path); however, since they are not central concepts, they are not being considered at present. The implications of the memory sharing permitted in Mach-1 and the inheritance of both memory and ports rights upon task creation have not yet been considered, and may also significantly impact issues of trust; the same is true for resource management approaches. This paper presents initial findings and strategies of research in progress, not the results of a completed research effort.

IV. KERNEL MEDIATION APPROACH

B3 criteria require sensitivity levels associated with all subjects and objects in the system and mediation of all access of subjects to objects based upon these labels. Subjects are active entities corresponding to users. In the Accent system, processes assume the role of subject, and as such should be labeled.

Objects are the passive entities and in traditional systems correspond to memory objects (data elements). In Accent, each process has its own virtual address space to define its virtual memory. This virtual address space has meaning only with respect to its associated process and is accessed only by that process. Since the virtual address space is so intimately associated with the process, and accessible only within the process (or in conjunction with the process's kernel port), it should be considered an indivisible part of the process and be labeled by the process label. This implies that the entirety of a virtual address space is at the same sensitivity level as that of its process.

Ports provide a unifying abstraction for the design of the Accent kernel and a focus for access control in the system. Communication between processes (request for services from the kernel and other servers, and the transfer of information) is accomplished by sending messages to ports and receiving messages from ports. The actual ports, or message queues, are maintained and protected by the kernel. Since ports provide the communication conduit and the access mechanism to process objects (and to their associated data and services) within Accent, and are not uniquely associated with processes (since ownership rights to them can be transferred), ports should be labeled.

Labels should be associated with ports and processes in the Accent kernel. Since the primary structures used to define and maintain both processes and ports exist internal to the kernel, these are accessible only to the kernel process and are protected from tampering by other processes. Initial examination suggests that the label associated with a process be stored in the PCBHandle and the port label in PortRec. Labels will require two fields: SecurityClassification and SecurityCategory.

At the time the user would log onto the trusted system, the Authorization Server would determine the values of these fields, based on the user's requested sensitivity level for the session. The validity of the request would be determined by the Authorization Server, based on the user's maximum sensitivity level as recorded in the Authorization Server's data structures. The Authorization Server would be modified either to insert this information directly into the PCBHandle record or to supply the information for another procedure to modify the PCBHandle.

In the case of Port objects, the NewPort procedure would store the fields of the creating process's PCBHandle record into the PortRec of the new port. This information would have to be duplicated in the PortRec, since the port's ProcId becomes "INTRANSIT" (NPROC +1) when ownership and receiver access rights are passed between processes, eliminating any possibility of verifying that the process receiving port rights is permitted to do so.

Since enforcement of access policies would occur at the time Send, Receive, or Ownership privileges were granted, there would be no need to enforce these policies during message queueing or dequeuing as well, except when the message being sent included port access rights. The IPC routines, GiveSendRights, GiveReceive-Rights, and GiveOwnership, would have to be modified to check the label and classification fields of the PortRec against those in the PCBHandle record of the process acquiring the rights, to ensure that the acquiring process had a valid right to access.

Discretionary access control would be provided by the Authorization, Authentication, and Name Servers using access group membership and ACLs, as is currently done in Accent. We have not yet investigated I/O handling and window interfaces (currently residing within the Accent kernel for performance reasons). Both areas may require significant redesign to accommodate TCSEC constraints.

V. SERVER MEDIATION APPROACH

Significant attention has been devoted in the existing Accent system to issues of access control. Built on top of the Accent kernel, its control of access through the use of ports is a collection of system processes, or servers, which work together to provide access control of named objects. The Authentication Server is called during the login process to verify that the user/password combination is valid. The Authentication Server interacts with the Authorization Server to acquire the password associated with any given user, and the access group membership of the user. If the user/password match is valid, the Authentication Server 1) establishes a port associated with the user, 2) records the access group membership, and 3) returns the Name Server port to the user.

Access group membership is determined by the values of the primary access and secondary access groups to which the user is a member. Each primary group is uniquely associated with a single user. Secondary groups may have both users and other secondary groups as members. These records are kept by the Authorization Server, which determines access group membership at login by performing the union of the transitive closure of the secondary access group memberships.

When the user wishes to acquire files or services, he interacts with the Name Server. The Name Server provides a port for a named object only if the user process's access group membership (acquired via interaction of the Name Server with the Authentication Server) corresponds to the ACL associated with the requested named object. This mechanism of access groups and ACLs is adequate to support discretionary access control and can be extended, with the addition of labels, to support mandatory access control.

The mechanisms described above exist in the current Accent system. The same server interactions (with modifications) can be used as the basis for adding labels and mandatory access control. Labels can be associated with users and kept in data structures maintained by the Authorization Server. The current sensitivity level for a session would be established at login time; it could be less than maximum clearance level of the user. The Authentication Server would keep information on the session level along with the access group information of the user that it previously maintained. All processes initiated by the user would operate at session sensitivity level, but they would not have actual labels associated with process control data structures. The session level for the process, maintained by the Authentication Server, would suffice.

When a named object is created, a label would be associated with it based upon the session sensitivity level. The object and its label would be "registered" with the Name Server. When attempting to access the object, the Name Server would check the session level (via interaction with the Authentication Server) against the object label, in addition to an ACL check against access group membership of the user, to provide both mandatory and discretionary access checks.

If access is permitted, the Name Server would return port access to the specific object requested. If the transaction involves process and object at the same sensitivity level, port access rights to the object would be conveyed by the Name Server directly to the process. If the transaction involves process and object at different sensitivity levels, a Mediation Server would be introduced as an intermediary. The Mediation Server would be given actual port access rights, an object ID, and the associated process identity; the user process would be provided with the object ID. Actual access to the object would be made by the user process through the Mediation Server, using the object ID. (The user process would acquire a port for the Mediation Server from the Authentication Server during the login to the trusted Accent system). This mechanism would give the user access to the object but prevent the user process from possessing (and transferring) port access rights.

Children spawned by a process would be at the same sensitivity level as the parent process and could inherit access rights held by the parent without violating mandatory access controls. The Mediation Server would intervene and hold ports only for transactions between processes at differing sensitivity levels.

We conjecture that inserting the Mediation Server into the object access path for selected transactions should not affect system performance too severely.

VI. CONCLUSIONS

The Accent and Mach-1 operating system kernels are very well-designed, and appear to provide a solid base upon which to structure trusted versions. Both systems provide discretionary access control which would satisfy C2 level ratings. (Other C2 level constraints, such as auditing requirements, would require non-substantive modifications.) Our current investigations indicate that the Accent system could be modified to generate a viable B3 level operating system. The Kernel Mediation approach, with labeling of all subjects and objects and a minimized TCB, is a strong candidate for B3. Estimates of the performance of such a trusted version have not yet been made, but we conjecture that the penalties imposed by proposed modifications to the kernel should not be severe. Required modification to control I/O and window management may alter this performance prediction, however. The Server Mediation approach presents a somewhat weaker but still viable case for B3. Performance should not be affected too adversely by the mediation required on transactions that cross sensitivity levels.

The brevity of our study has not permitted detailed examining the generalization of the kernel mediation approach to the Mach-1 system, although we are convinced that it will necessitate more significant modifications than required in Accent and a broader interpretation of the TCSEC. Nevertheless, a trusted version of Mach-1 seems achievable.

We are embarking upon further investigation of both approaches, with Kernel Mediation being the focus with the highest priority, since it is the most fundamental. It appears that the memory sharing permitted in Mach-1 will necessitate a stronger concept of memory object (capable of having an associated label) than is needed in Accent. The Server Mediation approach has second priority. Trust requirements are likely to suggest modifications in data structures, algorithms for access control interpretation, and organization of functions within the various servers.

VII. ACKNOWLEDGMENTS

We wish to thank Richard Rashid of CMU, and members of his research group, for the information provided to us in technical discussions of the Accent and Mach-1 systems. Their assistance has been invaluable.

VII. REFERENCES

1. Barron, R.; Rashid, R.; Tevanian, Jr., A.; and Young, M., Mach-1, Kernel Interface Manual, Technical Report, Computer Science Department, Carnegie-Mellon University, 15 January 1986.

2. Barron, R.; Rashid, R.; Siegel, E.; Tevanian, Jr., A.; and Young, M., Melange: A Multiprocessor-Oriented Operating System and Environment, Technical Report, Carnegie-Mellon University.

3. Department of Defense Trusted Computer System Evaluation Criteria, 15 August 1983.

4. Jones, M. B.; Thompson, M. R.; and Rashid, R. F., Sesame: The Spice File System, Technical Report, Computer Science Department, Carnegie-Mellon University, August, 1984.

5. Rashid, R., and Robertson, G., "Accent: A Communication-Oriented Network Operating System Kernel," Proceedings, 8th Symposium on Operating System Principles, ACM, December, 1981.

6. Rashid, R., The Accent Kernel Interface Manual, Technical Report, Computer Science Department, Carnegie-Mellon University, September, 1981.

7. Accent Source Code

AN OVERVIEW OF THE DoD COMPUTER SECURITY RDT&E PROGRAM

Panel Chairman, Mr. Lawrence Castro
Chief of the Office of Research and Development
National Computer Security Center

The purpose of this panel is to inform the audience of the progress of and plans for the Research, Development, Testing, and Evaluation (RDT&E) efforts sponsored by the Department of Defense (DoD) Computer Security Program (CSP). The presentation is organized according to the five distinct areas of the R&D program: Secure Architecture, Secure Database Management Systems (DBMS's), Network Security, Modeling and Verification, and Aids to Evaluation.

The first part of the presentation will allow each panel member to describe the status of his area's current programs and new initiatives for FY87. Among the new initiatives to be described is the consolidated program for producing a multilevel secure workstation. The participating panel members from the three military service labs will describe the support they are providing to the CSP. Following this, the panel will entertain questions from the floor.

Panel Members:

Mr. Wayne Weingaertner, Office of Research and Development (R&D), National Computer Security Center (NCSC), **Secure Architecture**

Dr. John Campbell, Office of R&D, NCSC, **Secure DBMS**

Mr. George Stephens, Office of R&D, NCSC, **Network Security**

Dr. Sylvan Pinsky, Office of R&D, NCSC, **Modeling and Verification** and **Aids to Evaluation**

Mr. H. Lubbes, Space and Naval Warfare Systems Command (SPAWAR)

Mr. John Faust, Rome Air Development Center (RADC)

Mr. John Preusse, Army Communications and Electronic Command (CECOM)

THE STRATEGY

The goal of the DoD's CSP is to provide a quantum increase in the security available to the nation's automated information systems. To achieve this goal, the NCSC has a three-pronged strategy. The first major component of that strategy involves a massive retrofit of security features into existing systems. The emphasis here will be on raising all federal computers to a controlled access protection level (the C2 level in the **DoD Trusted Computer System Evaluation Criteria**) of trust or better by 1988. The second prong of this strategy seeks to foster widespread availability of systems through the verified design level (A1) by 1990. The third part of the strategy is to develop techniques to extend assurances well beyond A1 in order to offer adequate protection for our most sensitive applications.

Federal-level policy changes, new operational procedures, and an aggressive R&D program were required to effectively implement the strategy. The R&D program contributes to the first part of the strategy through a concentrated effort to enhance some existing systems. The Computer Security RDT&E Program provides the means to experiment with the efficacy of various enhancement options --functions or features that might be added through enhancement, such as providing authentication, labeling, or auditing. With respect to the second portion of the strategy, the R&D program should continue to provide the technological support necessary in achieving an A1 system. Areas of support include stabilizing and improving verification environments; providing background material for refining security criteria --particularlyfor networks; refining security models that would serve as the point of departure in the development of A1 systems; and finally, developing A1 demonstration systems themselves. In accomplishing the third portion of the strategy -- going beyond A1 and transferring research breakthroughs into marketable products --the entire burden falls on the RDT&E Program.

THE RESOURCES

The Computer Security RDT&E Program is a cooperative undertaking led by the NCSC with the full participation of the Army, Navy, Air Force, Defense Communications Agency, and Defense Intelligence Agency.

Beginning with the FY84 budget, DoD RDT&E funds for computer security were consolidated, allowing the program build to be centralized while permitting decentralized execution. The FY86 program represents the third year of consolidation and, like the two before it, provides specifically-identified funds to be executed by several DoD components. Consolidation, as prescribed in DoD Directive 5215.1, avoids unnecessary duplication among DoD components. Decentralized execution of the program by the DoDCSC and the DoD components takes full advantage of the scarce expertise needed to provide technical oversight of contracts dealing with the highly technical field of computer security.

THE PROGRAM

To most effectively meet our challenge of transferring research breakthroughs into marketable products, we have channeled our efforts into the five distinct areas already mentioned. These five subprograms explore particular aspects of computer security research and development and, when combined, provide a solid program spiraling past the state of the art and into new technological frontiers.

Secure Architecture addresses the design and implementation of trusted computing bases (TCB's). A TCB is the hardware and software mechanism within a computer system that enforces security. Our current thrust is to push the edge of technology for TCB's. In addition, we are investigating kernel-based systems, office automation and personal computers (PC's), security enhancement of current systems, and advanced architecture.

Security kernels are the classical means of providing security in a TCB. A security kernel is a portion of the operating system that runs in its own domain, separate from the normal operating system code, intercepting any operation that has security relevance. Last year, Honeywell's kernel-based Secure Communications Processor (SCOMP) was successfully evaluated and received the Center's highest rating.

The prolific growth of office automation and PC equipment and software within the Federal Government is another area of research concern. Little consideration has been given to the security aspect of these stand-alone and netted office automated systems. Non-secure PC's, for example, negate the security provided by even the highest-rated host because labels used within secure computers that indicate the security level of the data are lost once data is transferred to a PC. Security enhancement will be targeted at next generation PC's since many of the current generation PC's are single-state machines and cannot support security.

Providing security enhancement of existing commercial operating systems that process classified information at inadequate security levels is a near-term solution. Under this task, we are incorporating security into the UNIX System V.

Advanced security architecture work provides new and different architectures for secure computers. The current effort in this area is the Secure Ada Target (SAT). The SAT takes a novel approach towards providing security in that it incorporates a separate security processor. Placing the security mechanism into a separate processor has notable advantages over the kernel-based approach. Because its architecture shares security-related portions of the system with nonsecurity-related parts, the kernelis open to attack. A separate security processor, however, prevents a user process from accessing the security-relevant portions of the system. A favorable side effect of security processors is an improvement in performance because it removes the security processing load from the main processor. This advanced architecture has completed initial design phase, and a prototype of this computer should be available in 1988.

Multilevel database management security R&D has received far less attention than has secure operating systems. In the summer of 1982, the Air Force and the National Science Foundation cohosted a workshop of experts in DBMS to examine the security problem. Three recommendations resulted: (1) provide near-term relief -- it is desperately needed and achievable; (2) for the mid-term, develop working demonstrations of high-leverage applications; and (3) conduct long-term research in the theoretical and practical foundations of secure multilevel DBMS's. Current and planned programs have made some progress towards achieving these goals, but there has been no breakthrough that substantially improves DBMS security.

The focus of the Secure DBMS subprogram is on compromise and integrity protection to databases and their related components. This subprogram is comprised of three research areas: trusted prototypes, studies and analyses, and advanced DBMS architectures. An effort to secure an existing DBMS entitled MISTRESS is now under way. Researchers are conducting various DBMS studies and analyses with the following objectives: data dependencies -- to achieve a family of multilevel secure DBMS's; evaluation -- to investigate the evaluation ramifications of DBMS's; and sanitization -- to examine the downgrading and upgrading of multilevel data in database systems.

A study is being conducted of the integrity lock technique, which cryptographically seals information stored in an automated system, with the objective of incorporating this technique directly into computer architectures supporting multilevel secure DBMS operations. And finally, the SAT will be used to develop a trusted DBMS application.

Network Security focuses on the protection of data while it is being transmitted between host computers and users. A data communications environment has been created between geographically dispersed computers that includes networks of computers, terminals attached to computers that are attached to networks, and the internetting of multiple and various combinations of these. Current computer networking technology has concentrated on providing services in a benign environment, and the security threats to these networks have been largely ignored. While literature abounds with examples of hackers wreaking havoc through access to public networks and the computers connected to them, hackers have exploited only a fraction of the vulnerabilities that exist. Techniques need to be developed that will prevent both passive exploitation (eavesdropping) and active exploitation (alteration of messages or message routing).

To reduce these vulnerabilities, we have initiated research in the development of components, high-level applications such as distributed processing, multilevel mail and file transfer, modeling, and advanced architectures. Within the area of advanced architectures, we are conducting internet research, device authentication studies, and architectural simulation. The challenges facing us in the network security field are boundless. We hope that coordinating efforts within the Federal Government and following a sound R&D program will enable us to work with industry to create a product line of network security systems that meet the needs of the Federal Government and will be available in the marketplace.

The problems of introducing computer security into the Ada programming langauge are being investigated. Ada is the DoD-mandated programming language for mission-critical systems. We are developing verification environments to be integrated into Ada software development systems as well as a suite of secure protocols in Ada to demonstrate how to marry these two technologies.

Our Aids to Evaluation subprogram addresses the need to streamline and improve the system evaluation process. We believe we can make the evaluation process more responsive to our national demand for computer security by providing a framework for identifying security requirements throughout the system's life cycle, identifying bottlenecks, automating tools to simplify the evaluation process, evaluating the effectiveness of safeguards, and reducing subjectivity in risk assessment. We are involved in research on intrusion detection, evaluation tools and techniques, erasure and emergency destruction, risk management, and generic product evaluation.

Modeling and Verification explores conceptual solutions to computer security problems (modeling) and provides assurance that system specifications and/or implementations are consistent with the model (verification). Research and development in modeling and verification addresses a critical national need for trusted software and hardware systems of high reliability. To extend the state of the art in security modeling and verification approaches, we have embarked on five research endeavors: Ada verification, integrated design and verification environment, security modeling, software verification, and hardware and firmware verification. Our ultimate research goal is to verify systems at all levels of design and implementation. In this regard, we note the similarity between our requirements and those of the Strategic Defense Initiative (SDI). We are working with the SDI program office to explore these common needs.

We believe our generic Computer Security R&D Program provides the solid framework needed to convert research breakthroughs into viable products. If you in industry, as the practitioners and managers of security technologies, think you can contribute to our efforts, we would like to hear from you.

COMPUTER ARCHITECTURES AND DATABASE SECURITY

Ronda R. Henning
Swen A. Walker

National Computer Security Center
Attn: Office of Research and Development
9800 Savage Road
Fort George G. Meade, MD 20755-6000
(301)-859-4488

ABSTRACT

There are various hardware/software architectures that will support a database management system and its assorted applications. Among the more common are the general purpose operating system/database management system combination, the backend database machine implemented in hardware or software, and the distributed database management system on several hosts. The distinction between the security responsibilities of the database management system and the operating system is not well defined. The responsibilities of the database management system depend upon the security policy of the operating system. Both database management systems and operating systems can provide some data security to user applications accessing a database. The question is how to divide the security controls between the two.

This paper discusses the various system configurations which support database management systems and the security tradeoffs inherent in each. It also details some of the fundamental security requirements and functions of the database management system, and the operating system security features that a database management system could take advantage of to enhance its own security features. The paper concludes with a summary of the author's current thought on secure database management architectures and their potential for near term implementation.

GENERAL ARCHITECTURE OVERVIEW

Current operating systems technology makes a case for three generic system architecture types that can support a database management system. These types are:

a general purpose operating system

a database machine environment

a distributed processing system.

General Purpose Operating System

The general purpose operating system environment (figure 1) provides a series of general services to a wide variety of users and their applications. These services include a file system, input/output management, user authorization and authentication, and recovery procedures. This class of system can best be categorized by products such as Unix[tm1], VMS[tm2], MVS[tm3], and other widely used time sharing systems.

[1] Unix is a registered Trademark of Bell Laboratories

[2] VMS is a registered Trademark of the Digital Equipment Corporation.

[3] MVS is a registered Trademark of the International Business Machines Corp.

Within the general purpose operating system environment, there are two basic types of security policies enforced: those that provide some degree of discretionary access control, and those that provide mandatory access controls. The security controls of most general purpose operating systems are usually only of a discretionary nature; they do not protect the user from the potential compromise of his data by his associates. For example, if a user gives read access to his file to another user, there is no mechanism to prevent the second user from copying the file and granting access to other users.

Discretionary access control is defined by the Trusted Computer Systems Evaluation Criteria (The Criteria) (1) as providing a means of restricting access to object based on the identity of subjects and/or groups to which they belong. According to the Criteria, the controls are discretionary in the sense that a subject (user) with certain access permission is capable of passing that permission on to any other subject. In the context of this discussion, systems meeting the requirements for the C2-level of the Criteria are defined as discretionary access control computer systems. For example, a user with read/write access to a file can grant read/write access to that file to another user.

A secure database management system existing on an operating system with good discretionary access controls affords the user the ability to control authorized access to the database on a per user basis. The operating system support of discretionary access control provides a higher degree of confidence in this protection because it is enforced not only by the database management system but also by the operating system's discretionary access control policy. This type of security is also most easily incorporated into existing products in both the operating systems and database management systems commercially available today. The general purpose operating system can also provide the database management system with authentication and authorization mechanisms. For example, the password generation and one-way encryption features of system login could also be used if a database were password protected.

Relying only upon a discretionary security policy to provide a foundation for a secure database management system may have serious disadvantages. Most currently available systems with a discretionary access control policy implemented can be easily circumvented. This allows a user to bypass the database management system's security controls and access any database directly from the operating system. Such an attack would allow the database files to be read with conventional file access techniques supported by any programming language. These systems are currently vulnerable to Trojan Horse penetration attacks because the user does not maintain complete control over the access rights to a file. For example, if a user runs an untrusted program, once this program grants read privileges to a malicious user, the second user cannot be prevented from granting read privileges to yet another user. Indeed, the entire user population could eventually gain read access to the file, thereby defeating the purpose of discretionary access control.

Discretionary access control operating systems may not have an automatic label enforcement mechanism to label data and files with the appropriate sensitivity marking. While these operating systems do provide some protection for the database management system, they do not promote a high degree of confidence in their controls, and, in effect, permit the user to define trust and thereby grant other users the ability to potentially compromise the data. Future implementations of discretionary access controls may provide better protection and enforce different security policies which may better address some of the shortcomings of current discretionary access control implementations.

General purpose operating systems with mandatory access control security policies implemented are usually multilevel secure systems. Mandatory access control is defined by the Criteria as a means of restricting access to objects based on the sensitivity (as represented by a label) of the information contained in the objects and the formal authorization or clearance of subjects to access information of such sensitivity. This means that users are protected from each other by the reference monitor using their sensitivity levels, which are used to label their processes, and by the labels attached to their files. Manipulation of file information is based on a composite of the mandatory access control and the discretionary access control labels. For example, a user cannot exist at the top secret level and modify an unclassified file, even if he has discretionary write access to the file. Such a "write-down" could violate the mandatory security policy of the system. Multilevel systems are relatively uncommon, the most notable being Honeywell's Multics system and the SCOMP (2).

A multilevel computer system can easily be thought of as a series of single level computer systems residing on the same hardware base and executing the same copy of the operating system at the same time with different processes at various levels cooperating with each other under the control of the operating system (3). Operating systems which enforce mandatory access control policies afford the database management system all of the advantages of discretionary access controls, but also add further security controls. Labeling, for example, is enforced on all objects (files and directories) known to the operating system.

Most operating systems with mandatory access controls are implemented with a security kernel architecture which enforces the system security policy on all access and authorization commands. The security kernel operates through a trusted path which allows the user to communicate directly to the trusted computing base during these operations. There are no such trusted paths or security kernels required in discretionary access control operating systems below the B2-level of the Criteria. The possibility of a system-wide Trojan Horse attack is also greatly minimized in a mandatory access control system because data is partitioned according to a user's authorization rights. A Trojan Horse attack would be limited to at most one sensitivity level of the system since objects are labeled by level and are not changeable by an untrusted process. This holds true only for disclosure of information, not data integrity. The existence of the trusted path also minimizes the risk of spoofing penetration because of the direct communication required between the user and the security kernel. All of these features lead to a higher degree of assurance that the security policy is enforced. That is, the user is protected against other users and maintains control over his data.

Mandatory security access controls do not solve all operating system security problems. The probability of covert storage and timing channels is greatly increased in comparison to discretionary access control systems by virtue of the fact that information must somehow be communicated between the levels for the operating system to function (4). Such channels offer a ready method of system exploitation if two or more users coordinate in an attack with cooperating processes.

Current implementations of security policy models do not provide sufficient

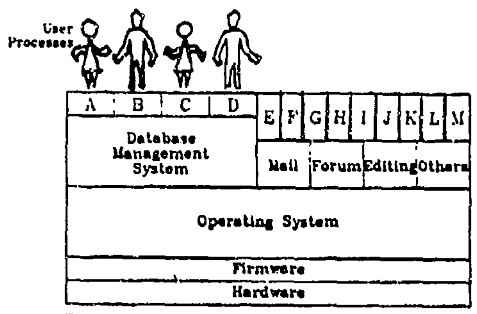


Fig.1 – GENERAL PURPOSE OPERATING SYSTEM

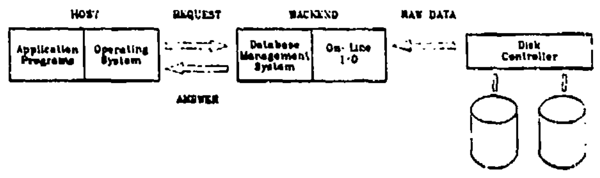


Fig.2 – GENERIC HARDWARE BACKEND DATABASE MACHINE

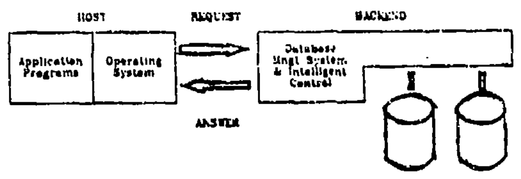


Fig.3 – HARDWARE BACKEND – HOST INDEPENDENT

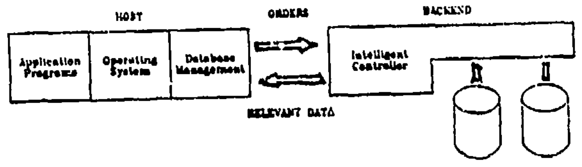


Fig.4 – HARDWARE BACKEND – HOST DEPENDENT

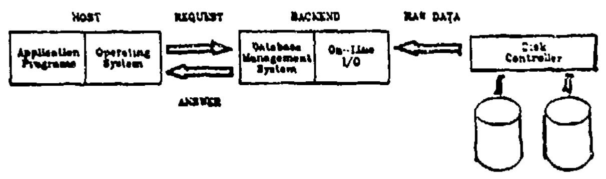


Fig.5 – SOFTWARE BACKEND DATABASE MACHINE

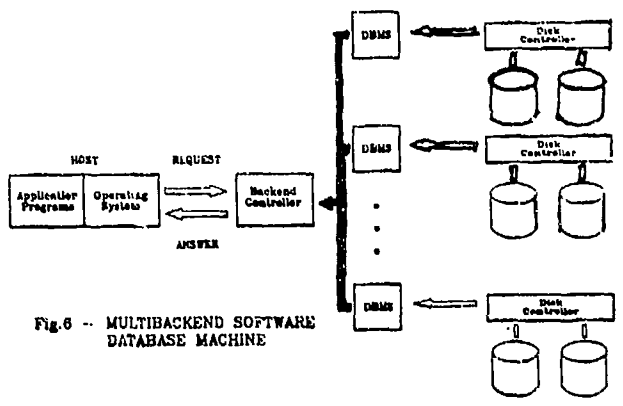


Fig.6 – MULTIBACKEND SOFTWARE DATABASE MACHINE

support for finer levels of label granularity. Sensitivity labels may not be automatically enforced to the level of granularity required for them to be used effectively in database management systems. For example, the operating system may support mandatory access control on objects down to the file level. Labeling on tuples or relations within a file, however, would be the domain of the database management system. There may also be severe performance penalties in that the general purpose operating system kernel and the database management system kernel have to communicate with each other to determine division of labor and perform file and directory manipulation. Some degree of confidence would also have to be assigned to the database management system in order for it to support its own security policy.

No database management system has yet been implemented that takes advantage of the multilevel features of general purpose mandatory access control operating systems. If such a database management system did exist, it could afford greater protection to the user and minimize the number of penetration techniques that could be used to exploit its flaws to a more finite set.

Database Machines

Backend database machines (figure 2) effectively remove the responsibilities for data access methods from the general purpose operating system. The backend database machine approach has been discussed for several years, but has become commercially available only within the last five years. This architecture is implemented in two basic configurations, the hardware and software backend database machines.

The hardware backend can be defined as a series of one or more processors which implement a particular database management system or type of database management system in custom firmware (5). These systems have been developed with two basic architectures; the standalone, host independent database machine and the intelligent disk controller. The host independent database machine (figure 3) encompasses all features of a database management system through a combination of its hardware and its operating system. In this architecture, the database machine must take all responsibilities for its own security and does not have the security protection of a general purpose operating system. The system can do all query processing and data reporting as if it were a general purpose operating system with the database management software in execution. Or the system can take requests passed to it from other frontend general purpose hosts and pass the results back to the requesting process.

The primary security advantage gained in this approach is the physical separation of the database management system from the frontend computer systems. This permits the database management system to control all access to the databases. The database machine has all security responsibilities for its storage devices and may implement its own protection mechanisms. The only dependency between the host independent database machine and the host is that of a channel to pass information back to the host and authentication and query information to the database machine. As an additional security measure, the database machine could require its own user authentication mechanism.

The independent database machine approach also implies that the database management system used on the database machine will only have to be secured once, with minimal changes to host-resident front-end software, as opposed to securing multiple versions of the database management system on a per host basis. Such a system could also enhance system performance on the frontend systems because all the security controls for the database management system are done on the backend database machine. This approach also allows hosts trusted at different sensitivity levels to communicate with the backend database machine, which, in turn, would ensure that they are permitted to access data only at their authorized sensitivity level. This feature provides data segregation without replication of complete computer systems. It should be noted, however, that the database machine would have to be trusted at least to the highest level of trust of the attached hosts. For example, a database machine connected to C2 and B3 level hosts would have to be trusted to at least the B3 level.

The backend host independent database machine is susceptible to covert signalling channels between cooperating processes. A trusted path between the database machine and host must exist to prevent Trojan Horse attacks and spoofing of the database machine. The mechanism for trusting the database machine is evident; how to build a trusted channel between the two systems is a harder problem. Encryption techniques alone do not appear to provide adequate protection. Other safeguards must be incorporated to secure the channel. Of course, the database machine architecture was designed to enforce security since retrofit of the mechanisms could result in a major restructuring of the system architecture to remove host dependent features.

The host dependent backend database machine (figure 4) can be viewed as an intelligent disk controller. This approach to the backend database machine implements and executes the high-level data manipulation language on the frontend host and requires a substantial amount of host-resident software to validate queries before they are sent to the controller for processing. The controller executes the most efficient implementation of the query to collect the data requested, but does not have the wherewithal to do very much with it except pass it back to the requesting process. The security assurances afforded by this architecture are those associated with a general purpose operating system's security policy. Current implementations of security constraints on these systems reflect the security policy of the database management system working in the frontend and are usually discretionary access control mechanisms.

The primary advantage to this approach is in the area of system performance. Response time can be dramatically decreased by using efficient search algorithms implemented in the backend controller. It is possible that some discretionary and mandatory access control could be done by the intelligent controller as part of its query processing. Once again, the intelligent controller must support the highest host level of trust if it does any security enforcement tasks. The dependence on the host system for most of the security policy enforcement in this approach can also become a liability if the security policy of the host system does not afford user applications adequate protection. The host dependent backend machine also must depend on the host system for authentication and user identification functions. Since it has no direct contact with the user, it cannot query him for additional authorization and authentication information and must rely on the mechanisms in place on the host system. The backend host dependent database machine must be reconfigured for each host operating system. That is, the front-end software, to do all necessary host functions, must be modified on a per operating system basis. A trusted path between the disk controller and the host must also exist to prevent penetration attacks and spoofing that would circumvent the system security mechanisms.

In the software backend database management system (figure 5), the specialized retrieval and parallel processing architectures of the hardware backend database machines are simulated by special software instead. This backend may exist on the general purpose computer system. It is similar to the intelligent disk controller. Once again, the security policy of the software backend approach mirrors the security policy of the particular database management system in use on the system. The principal advantage of this approach is the enhanced performance capabilities. Additionally, all of the resources of the operating system are available to the database management system. As a result, the security policy of the operating system can be readily incorporated into the database management system. Because of the implementation approach used here, it may also be possible to permit the database management system to enforce an alternative security policy that could coexist with the operating system security policy. The heavy reliance of this type of backend on the operating system can also be considered a major liability. Exporting this architecture to other operating systems and enforcing the proper function of the security mechanisms on other operating systems are the primary disadvantages with this architecture.

There is also a multiprocessor multibackend version of this architecture (figure 6) that utilizes a number of identical backends each using a copy of the same software. The single software system as well as the multibackend software system does not involve modification of the system's hardware; rather they rely on innovative software techniques to do their processing. This approach permits a security kernel to exist in the traffic controller (a particular software program) that routes queries to each backend according to their sensitivity level. Performance may also be improved because multiple backends can be processing portions of the same query in parallel and perform some of the required security enforcement. Physical segregation of the data can also be enforced by storing only data with a particular sensitivity label on a given backend. The system also complicates of configuration management control because there must be multiple copies of the software running on the system to access data on each controller. The problems associated with this approach are those of the single software backend architecture, but they are compounded because multiple backends exist. Additionally, each backend consults with the other backends occasionally in the course of query processing, thereby creating a potentially very large covert channel between the backends.

<u>Distributed Database Management</u>

The third generic architecture, a distributed database management system, is a relatively new technology and poses new security problems. Date defines a distributed database management system as any system involving multiple sites connected together into some kind of communications network, in which a user (end user or application programmer) at any site, can access data stored at any site (6). There are two basic types of distributed systems: homogeneous and heterogeneous. The homogeneous distributed system is one in which the same version of the database management system software and possibly the same operating system is running at each site. A heterogeneous system may have different database management systems, operating systems, and processors present at each site. From a security standpoint, the distributed case becomes very complicated because not only does database management security have to be considered, but operating system security and network security features must also be taken into account. Because this area is so unknown, it is difficult to determine the combination of mandatory and discretionary access controls that would be needed to secure a distributed database management system. Distributed and decentralized database control is a very important and difficult research area. Processing decisions in distributed database management system can be based on incomplete and inaccurate information when information from other hosts is unavailable. Formulation of the global security policy for a distributed database management system is not presently well understood.

If a security policy could be formulated for a distributed database management system, it would be most advantageous. Data could be logically and possibly geographically distributed among a variety of hosts, each of which could control his own sensitive data and authenticate queries from the other nodes on the system. The additional processors of a distributed system could improve system performance in a very large database environment with massive processing requirements. Smaller database applica would probably experience a degradation in performance because of the overhead of transporting data across the nodes. The possibility of data replication in the distributed environment also provides a method for trusted recovery. If a node on the distributed system crashes, it could conceivably recover and return to service by copying its databases from another node. Also, the removal of a node from service would not necessarily impair the overall security integrity of the distributed system because the other nodes would remain intact.

Distributed systems also raise many new and potentially serious security concerns. Perhaps the largest security loophole is concurrency control and database modification. Locking in the distributed environment has to be done very carefully to avoid denial of service to nonlocal nodes which may be doing retrievals against a database while another user is doing updates. Maintenance of journals in the distributed case is also difficult and must deal with many of the same considerations as the concurrency control problem. The possibility of compromise increases when data is accessed over a distributed system, simply because the user now has access to more than one computer system available for penetration attempts. Denial of service attacks are harder to detect and differentiate from a normal database lock on another node or the time spent in network traffic. The preservation of label integrity and label recognition across the nodes of the distributed system must also be addressed. It is also possible that the problems associated with data inference and aggregation will become increasingly more complex as additional nodes are added to a distributed system. In addition to all of these problems, the issues of network security must be considered in the development of the distributed database management system.

OPERATING SYSTEM/ DATABASE MANAGEMENT SYSTEM DEPENDENCIES

Within the framework of these general architectures, there are certain dependencies between the database management system and the operating system that apply in all cases. Stonebraker outlined these dependencies in 1981 and concluded that operating systems were inefficient and not designed to accommodate database management systems, but did not address the security considerations involved (7). Even if the database management system provides all of its own support in a backend database machine environment, the functions generally performed by a multi-user operating system are performed in the database machine kernel and hardware. What are these functional dependencies, and what implications do they have for a secure database architecture?

File Management

Perhaps the largest functional dependency is that of file management. Most database management systems use the system file handling routines to do some input/output processing. Most database management systems use the file system to implement relations, one relation or one database per file. Over this file, the data views or subschemas are superimposed. The operating system is responsible for opening, closing, update, and read operations on the relation or database. Additionally, the operating system opens and closes the data dictionary which enforces the database structure and subschema hierarchy on the user's process.

Beyond these basic functions, the file management system may also be relied upon to perform access control checking on the relations or databases being used. Whether this takes the form of mandatory and/or discretionary access control is a direct function of the security policy implemented by the operating system. In the case of the general purpose operating system, discretionary access control on relations or databases can be readily enforced since they are already incorporated into some commercial products. Operating system discretionary access controls reinforce these existing mechanisms, but are dependent upon the storage structures of the database management system. If the user does not have write access to the file, for example, he cannot update a relation or database. If the user does not have add privileges, he may be able to update existing tuples but cannot create new tuples. In mandatory access control, the database management system also depends upon the operating system to validate the user and file authorization labels to ensure that the operation requested does not conflict with the system security policy. Reliance on system file management routines does present solutions to several areas of security concern. However, file input/output is not usually the most efficient access method for database management and operating system file management routines may be bypassed. To optimize database performance, smaller buffer sizes and blocks of data are preferred to manipulation of file-sized structures. This also leads to the issue of object label granularity, which is discussed elsewhere in this paper.

A special area of file management within the database management environment is that of recovery services and auditing. Most database management systems that keep journals or transaction logs exist in a single level environment. Those few that have tried to exist in a multilevel system have done so by maintaining separate journals at each level, making recovery procedures complex. The dilemma with journals is that they must be kept, which implies they have to be written to by each user. However, users should not be able to read what they have written to the journal, and its existence should be transparent to the user. This applies in most cases. In the event the user cannot commit a transaction to the database, however, the database management system should be able to initiate a rollback on the user's behalf and restart the transaction. The database management system should also be able to examine a database for damage in the event the system should crash and to restore the database management system and its accompanying databases to a consistent state.

Closely related to recovery and file management is audit maintenance and control. Many of the same problems inherent in transaction journals also exist in audit files. Indeed, in most systems, they are one and the same. However, the database management system can, and often does, use the system audit facilities to create a bare bones audit of its own. This type of audit contains only information related to tuples that updated the appearance of the database. Those tuples which are read out of the database for a report, for example, are not included in this type of audit. An audit such as this would be highly useful in recovery, but would not be very useful if the database administrator or system security officer was trying to determine if an inference or penetration attack was underway. Such an attack would not have to modify the data but only collect it. The problems of enforcing the appropriate access controls also continue to exist in audit functions and files as well.

Additionally, the question of what the database treats as a subject and what the database treats as an object for audit purposes must be answered. If the object for an audit is defined as each individual attribute in a database, the audit files will quickly become very unwieldy. If the object is the relation, on the other hand, not enough information will be available for a meaningful audit trail to exist. By the same token, it may make more sense in the database environment to audit by object than by subject. That is, the data accessed should be audited as opposed to the user or accessor of that data. In database management systems, one is more interested in access attempts on a particular sensitive data item than in all actions done by a particular user. Additionally, if the audit is conducted on a per attribute basis, auditing on a subject level could result in very large audit logs. These areas must be addressed if a database management system will manage data securely.

Label Granularity

The issue of label granularity is another area in which there are functional dependencies between the operating system and the database management system. Most operating systems only support labeling at the file and directory (a logical collection of files) level. They do not support the labeling of entities smaller than a file. Therefore, most database management systems have to maintain their own labels and be responsible for their integrity. Database management systems cannot use the system level mandatory and discretionary access control mechanisms to enforce access rights to any finer level of granularity. In fact, the database management system will probably have to violate the operating system security policy to manipulate smaller multilevel objects. This, in turn, causes concurrency control problems because locking can only be accomplished when access control can be enforced on the database. Additionally, the question of whether or not to trust labels which are not enforced by the operating system security kernel must be addressed. Another issue in this area is the interface between the database management system's labeling techniques and the operating system's labeling technique. In this case, should labels attached by the database management system be considered valid by the operating system? Current technology may not permit a lower level of labeling without a substantial performance penalty, rendering a database management system unusable in real-time user response time applications.

Memory Management

The database management system may rely on the operating system to perform memory management on its behalf as well. In this area the largest functional dependency is that of object reuse. The database management system uses those pages of memory the system allocates to it and leaves the system to perform object reuse functions on those pages. For example, if the page was not modified, it will not be written back out to disk. If the page was modified, it must be checked to ensure that the labels associated with that page are still valid before it is put back on the disk. Additionally, before the page frame is reused, it must be overwritten to prevent recovery of data by the next unauthorized user. Some operating systems leave object reuse/data remanence up to the particular application. Others perform the function as a matter of course. The database management system should make no assumption as to what services the operating system provides for it in this regard, but could take advantage of operating system services if they are available.

Denial of Service

The question of denial of service is also tied to a database management system's dependencies on the operating system. For example, if the operating system has decided to swap out a process that has the database locked for update, the competing processes which may wish to use that database are denied service until the swapped process is brought back in to finish its execution. With the possible exception of custom database machine environments, the case of denial of service in database management is complicated not only by the database management system's own locking algorithms, but by the operating system's process scheduler software. Synchronization in the interaction of the database management system and the scheduler may make an database management system too dependent on a particular operating system to make it portable to other operating systems.

Interrupt Handling

Connected to the memory management dependencies are the database management system's dependencies on the operating system in the area of interrupt handling. By necessity, database management systems generate interrupt requests to the operating system to perform various operations such as requests for files, input/output handling, and process wakeup and block messages. The operating system, in turn, executes these requests and generates the appropriate interrupts, which, in turn, may be audited by the operating system as needed. The interrupt interaction between the database management system and the operating system offers a wide range of opportunities for covert channel exploitation. For example, a user could have a database locked for exclusive update and then trigger an interrupt that would suspend his process. Another user would not be able to access the database while the lock table showed that the other user had it exclusively locked. In this case, the lock table could be used as a covert signalling channel between the two user's processes. Unfortunately, with the possible exception of the database machine case, there is little that can be done to minimize the need for such communication during interrupt processing, although the implementation of interrupt handling in a secure operating system may afford sufficient security features to minimize the opportunity for covert channel exploitation through the interrupt processing facilities.

Interrupt processing raises the question of the concept of the trusted path and trusted processes. Certain functions of an operating system, for example, the input/output controller and the mail system, have the ability to bypass system standard access control policies. These processes are known as trusted processes, and their ability to communicate with the security kernel requires a "trusted path" to the security primitives. Obviously, labels have to be enforced from the user perspective, but certain operations, such as label manipulation and modification, have to be considered trusted processes. Their use should be permitted only by database administrators or system security officers, and, even then, heavily audited. Obviously, for regular functions, the trusted process has to exist as a standard feature and be used rather frequently. In these cases, the trusted process resides at the highest security level accessible to the user. When a request that requires trusted intervention is made, the trusted process filters the request to the appropriate files, and consolidates the returned responses before passing them back to the user at the appropriate level. Once again, there is a substantial possibility of large covert channels, but extensive audit files and system security controls on user processes help to minimize the associated risks.

Authentication and Auditing

Related to trusted processes is the area of authentication and authorization techniques. The database management system may not require that each database is password protected. It may use the system's authorization control mechanism to enforce access control on the database. In turn, use of the authorization control implies that the database management system is willing to accept the authentication mechanisms of the operating system as well. With such a scenario, the user logs into the system, which in turn authenticates his identity and validates his authorization levels for the existence of his process. The database management system, in turn, uses the user's authenticated information to determine his access rights to the data. It is also possible for the database management system to take advantage of the operating system's enforcement mechanisms for mandatory and discretionary access control through the authentication/authorization mechanism. That is, if the user is not at the appropriate level in a multilevel system and attempts to access a database in violation of the system security policy, the operating system access mechanism may prevent this occurrence and generate an interrupt, which must be intercepted and interpreted by the database management system. The same scenario could also be used in the case of discretionary access control where the user attempts to update a relation he is only permitted to read. The complexity of the interface between the database management system and the operating system in this area could be reduced if the database management system did its own validation on access requests prior to passing these requests to the operating system, which would, in turn, minimize security-related interrupts. This dependency is a very necessary one if a trusted database management system is to coexist with a trusted operating system successfully.

Network Services

All of the above relationships between the database management system and the operating system have been general case dependencies. There is one very critical dependency that exists in the distributed database management environment, the network server function. In this case, the data and user requests are transmitted through the distributed system to any and all appropriate hosts, which in turn return the requested information to the sender. The network server should be a trusted process communicating with other trusted network servers who presumably reside on trusted operating systems. This model holds for either of the distributed architectures discussed above because either trusted queries are sent without the benefit of a trusted control system to determine where they should be routed, or with this benefit. In either case, the netserver function is a necessary dependency in a distributed database environment that will require further exploration before such an environment can be considered trusted.

DATABASE MANAGEMENT SECURITY FUNCTIONS

Beyond the functional dependencies between the operating system and database management systems, there are security functions which must be performed by the database management system independent of the operating system. Those features which have an effect on the database management system's architecture are discussed below.

Label Enforcement

Perhaps one of the most important security relevant functions of the database management system is that of label enforcement at fine granularities. Because current trusted operating systems may or may not recognize objects for labeling that are smaller than a file, the database management functions which must be performed by the database management system independent of the operating system. Those features which have an effect on the database management system's architecture are discussed below.

Label Enforcement

Perhaps one of the most important security relevant functions of the database management system is that of label enforcement at fine granularities. Because current trusted operating systems may or may not recognize objects for labeling that are smaller than a file, the database management system must take responsibility for labeling at lower levels. These levels, depending on the database architecture, are the relation, tuple and attribute. Some database management systems use one file per relation, therefore they are only concerned with label management at the tuple and attribute level and leave file level label management to the operating system. Assuming discretionary and mandatory access control information has to be accounted for, the database management system must account for the sensitivity level of the data as well as the access control lists attached to the data. For example, the sensitivity level may be top secret and the discretionary privileges may be read and update for a particular tuple. How low the level of label granularity applies depends on the security policy enforced by the database management system.

Label Integrity

How is label integrity maintained? The sensitivity level used in labeling can be ascertained from the user's process authorization. This label then has to be attached to all new information entered into the database by this process. Users should not be able to modify existing labels, nor should they have the ability to enter or change data at lower or higher authorizations than the one they currently are using. This interpretation of the multilevel environment is highly restrictive and most users would consider it to be a user-hostile denial of service. Label modification should only occur under the auspices of a trusted process. The database management system must automatically append the appropriate label to the data upon data entry into the database. Additionally, the database management system must do label comparison operations to determine if update, delete, and read operations follow the security constraints of the system. These functions must maintain label integrity to ensure correct operation of the system security mechanisms and maintain the system security policy. This enforcement is also an important consideration in the system integrity policy to prevent unintentional or malicious corruption of data.

Query Interpretation

Another area of security responsibility is that of query interpretation. The security mechanism of the database management system must ensure that what information the user requests via a query is what he receives, consistent with the security policy, of course. This means the integrity of the query must be beyond reproach. In fact, it may not be possible to provide this level of query integrity without trusting the entire database management system. There must be no chance for the insertion of Trojan Horses or violation of the system integrity policy before the query is processed. Additionally, safeguards must be in place to enforce the data labels and mandatory access control policy on the reply to the query. Once again, the issue of label integrity must be addressed if correct results are to be obtained from the database. Labels may be appended to queries to ensure their compliance with mandatory access controls, in which case the database management system is responsible for secure query modification that only appends the subset of labels appropriate to a given process and query and not the superset of all known labels which may reside in a given database. In the event query modification techniques are not employed, the database management system must perform a filter function before returning any information back to the user in response

appended to queries to ensure their compliance with mandatory access controls, in which case the database management system is responsible for secure query modification that only appends the subset of labels appropriate to a given process and query and not the superset of all known labels which may reside in a given database. In the event query modification techniques are not employed, the database management system must perform a filter function before returning any information back to the user in response

to a query. This filter, of course, must be a trusted process capable of examining all returned data and forwarding only that which the user is authorized to examine. This can be especially crucial in the case of an update request, where the user may be able to examine lower level data but can modify only those components of a tuple which are at his current authorization level. Under such circumstances, labeling of attributes and checking of authorization privileges are very much secure database management functions.

Indices

The use of indices as a mechanism to improve response time also causes security concerns. If the indices are derived directly from data, or if their use can reveal additional information about the database structure or contents, then their existence raises the same types of labeling and query modification concerns that apply to generic data. Indices also have to be sorted by mandatory access control level and used in a manner consistent with the database management system security policy. Therefore, what started out as a method to improve retrieval performance may actually hinder it by the time all necessary access mechanisms are enforced on the indices. In this case, use of the indices may become more burdensome and less efficient.

In the event data indices exist as the result of a random hash algorithm or other arbitrary addressing technique, the question of where the indices reside must be addressed. If they are placed at the highest authorization level, once again the data filter problem exists. If the indices are located at the user's lowest authorization level, they can be read by all but may unintentionally divulge information about higher level data and could be exploited in a statistical inference attack. Location of data indices and their use therefore becomes another security function for the database management system to balance between data security and system usability.

Data Dictionary Enforcement

Indices, however, are only a small segment of a larger security concern in database management, that is, how to enforce data dictionary constraints securely. The data dictionary contains the database schema, data conversion algorithms, and characteristics of data attributes. In most systems, the data dictionary is invoked for data validation whenever a query is posed or data is modified or added to the database. Since the data dictionary could contain data validity checks which might divulge sensitive information about data value ranges for a database, it should have associated with it a level of trust equal to the highest level of trust for a given database. The data dictionary also may contain information about discretionary access privileges and the location of database files which contain the actual data. Therefore, the data dictionary also becomes an active entity which requires protection beyond that which is normally available in untrusted operating systems.

Perhaps a solution is to divide the data dictionary into its components and protect each of these according to their relative sensitivities. Such a solution would have to be determined on a case by case basis for each individual database and data dictionary combination, thereby eliminating the possibility of a generic enforcement mechanism. Another solution would be to translate the data dictionary into an executable form which could be kept at a level accessible to all users while the original source from which it was derived resides at the highest sensitivity level under the further protection of discretionary access control administered by the database administrator or security officer. The executable form must be enforced as truly "execute only" permission. That is, the user must not be able to reconstruct the source segment nor determine sensitive algorithms or data. This method allows all users access to the information in the data dictionary while protecting the information as closely as possible. It strikes a compromise between total access and complete security while preserving the required functionality for the database management system. Such a compromise must be qualitatively measured for each trusted application according to the security constraints in force at that level of trust.

Database Creation and Security Functions

Data validation and retrieval performance aids, however, make one important assumption -- that the database exists and has been defined and created by a user of the system. This becomes a security concern because the data files and supporting structures such as the data dictionary and lock table should be known only to the system and the database administrator. They should not be accessible to the common user accessing data through the database management system. To make them accessible and known to the users by their appearance in the directory structure invites tampering. For example, if the system does mandatory and discretionary access control to the file level, and the database management system supports one file per relation and uses the system's access control mechanisms, there is nothing to force the user through the database management system to access the entire relation, whether he has access rights established on a per tuple basis or not. A simple copy or print command would result in the compromise of all data stored in such a relation, not to mention all per tuple privilege information for the relation. At this point, it is a small job to decipher the relation formation and complete the data compromise.

The problem becomes even more complex in the case of a multilevel operating system. Here the database manager somehow has to segregate data according to the user's authorized level, but merge it to provide responses to his requests consistent with the system's security policy. That is, the data has to be stored in such a way as to maintain the mandatory access control policy of the system while providing the user with his information. The database management system has to maintain the appropriate data files at the correct levels, or store the data at the user's highest authorization and distribute it from there.

The majority of work on multilevel data management to date has assumed the database existed and worked from there. Very little has been done on the potential covert channels that might be created during the initialization of a multilevel database, or techniques to minimize them.

View Enforcement

Related to the problems of database creation is the area of view or subschema enforcement. This function enforces certain configurations of the basic database as created by the database administrator on the database's users. Historically, views have been defined in the data dictionary/database creation files. Their enforcement upon the database becomes critical in multilevel data management. For example, the database as a whole could exist at a variety of levels with views used to ensure that the user only sees data consistent with his authorization level. Views can also be used to enforce discretionary access to the database if one view is used for each discretionary access right and composite views can exist to allow users multiple privileges. For example, separate views exist for reading and writing data to a particular relation and the two views are merged to create a read/write view for a particular user. Once again, if the database's data files are accessible with the standard system file commands, views are easy to circumvent. Additionally, combinations of views may divulge more information than single views authorize the user to see. View mechanisms may also be used to protect a database against inference attacks and to ensure data integrity. In light of these potential security uses of views, their secure enforcement becomes a prominent security service that must be trusted to work correctly.

Trusted Process Mechanism

As was noted above in the discussion of operating system dependencies, creation and use of a trusted process mechanism in the database management system can become a major security concern. In this context the trusted process becomes responsible for the enforcement of the database security policy and ensures its consistency with the operating system's security policy. It is this process which takes the user's request and applies appropriate logic to it to result in an updated view of the database. If a user were to request an update of a tuple in the database, the trusted process would be responsible for validating his access privileges on the relation and on each attribute of the relation, if necessary. It would also be responsible for ensuring the integrity of the database, that is, checking the lock table prior to applying a transaction to the database to ensure each user gets consistent information. The trusted processes become even more critical in the multilevel environment. Here they must still ensure that data integrity constraints and locking protocols are followed as well as handle the multilevel security policy axioms. In the case of the database machine, these processes are the primary components of the security kernel. A database management system working in conjunction with a trusted operating system would interface its trusted processes to the operating systems's security kernel, creating a large opportunity for covert channels and subsequent system exploitation. In the event the system permits execution of user data segments, this exploitation threat is substantial.

Auditing Small Objects

Another requirement to maintain data integrity is the need for auditing at a finer level of object granularity than the file level objects most general purpose trusted systems audit. The majority of operating system audit tools would note that a user accessed or attempted to access a file and may or may not have modified it. Special audit tools used for system debugging account for arguments passed into and out of subroutines that are referenced. In the case of secure database management, a combination of these two functions is required. An audit that notes only that the user accessed a database is not sufficient to address data security concerns. For a database management system, a useful audit function would probably include the data accessed and the before and after image of any data modified by a user request. The audit log also has to be able to account for the possibility of multilevel data manipulation by either existing at each level or at the user's highest authorization. In this case the audit log will grow rapidly into something very large with minimal utility. Therefore, for a trusted audit to exist successfully in a database management environment audit reduction tools must exist, including pattern recognition tools that could detect attempted inference attacks. Both types of tools should exist, otherwise valuable pattern information could be lost during audit reduction and an attack could occur and never be detected through the audit logs.

Such an audit log could also be used to handle trusted database recovery as well. Recovery for a database management system becomes a bit more complicated than recovery for a generic operating system. The operating system has to save its hardware context (the values of its registers at the time of crash) and write all modified pages back out to disk. It makes no claims with regard to data validity within those pages, and, if it cannot get the page back on disk, may replace it with a page of null values. This type of nonrecoverable error would be easily detected in the case of an individual text file, for example, but a database can be much larger than a single page and therefore such an error could conceivably escape detection until data from that particular page was needed and not found. The database recovery facility must be able to determine if its files were affected by a crash, if pages within a file were affected, and, if so, to restore the appropriate information. The information from the audit logs can be used to determine if transactions have been committed and to repair damaged databases. In trusted mandatory access control systems, the integrity of data labels must also be validated. Additionally, if the system is multilevel secure then the data must be distributed to each level securely. The recovery mechanism may have to deny service to database users while it is validating the database after the system has finished its own recovery to ensure as accurate a reconstruction of the database as is possible. This technique requires that the database recovery manager must be added to the trusted processes resident in the database management system.

Inference and Integrity

No discussion of database security mechanisms would be complete without the mention of data inference and integrity control mechanisms. Data inference -- the unintentional compromise by deduction of unauthorized information due to combinations of the possession, known existence, known absence, chronology, and location of authorized information -- is most frequently exploitable in data at either end of a standard distribution. That is, the most extreme values are the most vulnerable. There are several techniques to protect against inference attacks, but the majority of them render the database useless for precisely formulated queries with specific responses because they involve the corruption of the original data in some way. An alternative approach to inference control is the construction of a rule-based semantic layer between the logical database design and the physical implementation of that schema. This rule-based system would use statistical information about the database composition to determine the probability of compromise if the requested information were divulged. If the probability of compromise were high, the data would not be revealed. It is possible, however, that the performance penalties paid for inference control may make a database management system unusable in an interactive system.

The concerns of data integrity are more acute and offer more promising near-term solutions than those of data inference. Data integrity in the database management sense is defined as the correctness of the data itself and any associated data structures and information required to access the database. The principal concerns in the area of database integrity are associated with locking mechanisms for the update and addition of information to a database. If a user is updating the database, an exclusive lock mechanism must deny other users attempting to update the database or retrieve information access for the duration of the update. It may be possible to preserve uncommitted transactions and apply them to a database when the locking process releases its locks, thereby freeing the database for other users. However, there may be complications with this strategy in that there is no guarantee that conflicting transactions would not be applied to the same data. For example, two users wish to modify status information and attempt to commit transactions at the same time. One may say the status is complete, the other the status is pending. Such types of conflict cannot be resolved automatically by the system and would require human intervention of some sort. The update problem becomes more complicated in the multilevel sense in that users at different authorizations could well be modifying attributes of the same tuple at the same time. The database management system must apply the transactions as specified by the security policy of the operating system. These integrity constraints do not include the case of unauthorized intentional modification of data by an authorized users. In this case, a user modifies a file at a higher classification by writing up into it, although he cannot read the file afterwards from his current authorization level. There is no known security policy that addresses this concern and maintains a multilevel user environment.

There are integrity concerns with the locking mechanism. The lock table may not reflect the current status of the database. For example, if the system crashed after the user process committed a transaction but before he could release his lock on the database, denial of service to other users could be based on incorrect information because the lock table is left in an inconsistent state. Given the option, the database recovery manager could possibly resolve a inconsistent lock table from the audit logs. It could not resolve an inconsistent database from a consistent lock table in most cases.

Denial of Service

Beyond these integrity issues, there are the questions of denial of service in a multilevel environment. A user at a higher authorization level could have the database exclusively locked. This fact must be hidden from the lower level user t prevent a covert signalling channel. However, the lower level user could not access the database if the higher level user was working with a particular page. It has been proposed that creating mirror image tuples at the appropriate levels would solve this problem; however, the question then becomes which user has the most current version of the tuple and which user is working with data that has been modified without his knowledge.

The above concerns are only meant to highlight the severity and importance of the database management system's security functions. They are not meant to be complete discussions of the subjects, but rather to show the magnitude and impact of the security constraints which will exist in a trusted database management system.

EXPLORATORY DATABASE ARCHITECTURES

Given the number of dependencies between the operating system and the database management system, the security functions the database management system must perform, and the state of current technology, can anything be done to minimize the security problems in database management? The majority of currently available database management systems address the discretionary access control constraints in some fashion, even if they are easy to compromise by experienced programmers. These discretionary access controls do perform the function of protecting the user and the database from unintentional mistakes that could cause data leakage. They do not protect against deliberate attacks. The few database management systems that are hosted on multilevel systems do work well with the mandatory access controls, but they do not support multilevel objects and cannot function at more than a single level per database.

What is needed then, is a method to enforce discretionary access controls securely, force all access to the database through the database management system to eliminate the possibility of copying data files through the file system commands, enforce mandatory access control, and work in a multilevel environment without creating large covert channels. To do so efficiently with minimal impact on user response time is a necessary condition if the product will be usable. The question then becomes how to meet all of these requirements in a database management system that can be implemented in the near future. The 1982 Summer Study on Multilevel Data Management proposed three different architectures to answer these requirements (8). This section summarizes these three architectures and includes a fourth architecture that the authors believe may offer a solution.

Kernel-Kernel

The first of these architectures, the kernel-kernel approach, uses the operating system security kernel as a foundation for a database security kernel that acts as a trusted mediator between the user's requests and the operating system security kernel. The database kernel is responsible for labeling at a granularity finer than the operating system's smallest labeled object. The operating system does labeling at its granularity levels and is responsible for the enforcement of the system security policy on the database management system. The operating system ensures that the user can only modify data at his current authorization level through mandatory access controls. The database management system attaches the appropriate labels to the data and enforces its discretionary access controls on its databases. Recovery operations are a joint effort between the database recovery manager and the operating system's recovery utilities. The operating system recovers to the file level and informs the database recovery manager that there was damage done to items under its control. The database recovery manager then examines its databases and does what it can to make things consistent. Comprehensive system audit tools are customized to handle the database audit requirements by adding audit reduction and pattern recognition features. Authentication of the user is done by the operating system and the database management system may take advantage of that information or use the system authentication subroutines to perform its own authentication.

The kernel-kernel approach should allow retrofit of the database security kernel on a kernelized trusted operating system. The primary area of concern in this approach is the definition of the boundaries of the database security kernel. In the worst case, the databases themselves must be within the bounds of the kernel, making it so large and complex that correct operation could not be substantiated. In the best case, the kernel may not be substantially larger than the operating system security kernel and would have little effect on performance or validity.

This design may take advantage of the security features of the underlying operating system for mandatory and/or discretionary access control. This technique could conceivably be applied to any database management system that resides on a kernelized operating system host. Performance constraints on the database management system would exist if the performance of the operating system security kernel was poor since it must interact with the database kernal for most operations. Covert channels may exist because of the interaction between the two kernels. Such covert channels also increase the probability of Trojan Horse attacks against the database management system by cooperating processes at various sensitivity levels. Additionally, the no-write-down constraint of the Bell-LaPadula security model prevents data from being stored at a sensitivity below the current authorized sensitivity level of the user, making this system very user-hostile for data update operations. To minimize the user interface problems, larger covert channels would have to be permitted and a generic downgrade function would have to exist in the database management system trusted software and be accessible to authorized database users. This approach is very attractive in that the amount of trusted code to be added to the system is relatively small because the database management system uses the operating system for the majority of its trusted functions, making it easy to retrofit into an existing system. However, the user interface to this database management system, and the potentially large covert channels may not make it useful as more than a demonstration project. Only if past experience with kernelized architecture performance constraints can be incorporated into the system design would such an architecture be feasible for secure database management systems.

Cryptographic Sealing

The second exploratory architecture, cryptographic sealing, uses encrypted checksums to determine the authorization label and access rights to the data. When a tuple is created, the encrypted access information is appended to it. Every time the data is accessed, the sensitivity labels are decrypted with keys corresponding to each access class. If the data is decrypted proper, it is forwarded to the user. If the correct key is not located, the data is not returned. A variation on this method uses query modification to append the correct sensitivity label to the query and stores the labels as another attribute with the rest of the data. The label fields are then compared as part of the normal query/response processing with matching labels required before an item is reported to the user. In these examples, there is additional overhead for encryption/decryption and query modification. The database management system is responsible for all label integrity and access control enforcement. Recovery beyond the file level is handled in much the same way as the kernel-kernel approach with the database recovery manager responsible for label integrity and data correction if necessary.

The principal disadvantage to this method is the time involved to encrypt and decrypt the labels and the additional storage required for them, since sensitivity labels are not usually stored with data. Sensitivity keys must also be stored with care so they may not compromise the system's security mechanisms. There is also a possibility of compromise through unintentional or intentional mismanagement of the encryption keys and checksums. The advantage to this method is that the database management system becomes responsible for all access control functions and performs as a simple access filter. This is especially true in the variation on this technique that encrypts the entire tuple and decrypts it only when necessary. The fact that the only trusted component in the system is the filter makes it simpler to verify correct operation of the software and a relatively straightforward approach. Potentially, this system offers a high degree of trust and can be incorporated into an existing database management system with minimal effort, if the constraints involved in key management can be resolved.

Physical Data Segregation (Backend Database Machine)

A third approach, that of physical data segregation, is best suited to the dedicated database machine environment. In this method the mandatory access control is accomplished by independent processors and disks which are labeled by sensitivity level. Each independent processor works on a query passed to it by a central controller and returns the requested information under its control that meets the conditions of the query. The control processor determines which independent processors to forward the query to and merges the independent responses into one consolidated response. The independent processors are responsible for discretionary access control on their own data and the control processor is responsible for mandatory access control enforcement.

One of the disadvantages of this approach is the extra complement of processors and disks associated with each level. Additionally, each device pair must be responsible for its own recovery and auditing functions. This method also has to cope with components of the distributed database locking problem in that it may be able to obtain locks for high level data but not for low level data, resulting in denial of service to a process which requires both levels to develop an answer.

The major advantage is that the security controls are relatively centralized in the control processor, thereby defining the bounds of the database security kernel and its interfaces to nontrusted processes. This technique offers a straightforward approach to secure database management. However, if too many features are incorporated into the front-end controller, the security constraints may become very complicated.

Custom Kernel

The fourth primary architectural alternative is a operating system security kernel designed with database security features in mind. This type of architecture cannot be incorporated into a system, it has to be designed in and exist from the start. The time usually spent determining where to place security constraints is instead spent on design of the system from scratch. The implementation allows the database security policy to be reflected in the system security policy. The operating system can take responsibility for all labeling functions and the enforcement of the mandatory and discretionary access control policies. The database management system is responsible for additional security requirements such as inference control. Audit functions can be handled by the system audit controls since the system recognizes labels on objects smaller than a file in this scenario. Recovery procedures could be managed by the operating system since it understands the smaller granularity and the labels associated with it.

This technique may also prove very costly in that it requires the expansion of the security kernel to incorporate the database security kernel's functions. There may also be performance problems that would render the system user-hostile in interactive environments. Past experiences with large operating system kernels have demonstrated that system performance and the size of the kernel are related, with large kernels being harder to validate and slower (9). In this context, much of the validation information and audit functions would probably have to be implemented in hardware to ensure adequate response time for the user's applications. This approach would only be feasible if the operating system kernel could be extended without compromising its' level of trust or its' ability to be analyzed for security flaws. Therefore a dedicated database machine architecture is implied because the system would be tailored to address database security considerations.

CONCLUSIONS

There are several functional dependencies between the database management system's security functions and the operating system's security functions. These dependencies would make it very difficult if not impossible to develop a trusted database management system on an untrusted operating system. They would also make it hard to trust a database management system beyond the level of trust available in the operating system. For example, it would be difficult for a database management system to enforce strong mandatory access controls without the underlying support of an operating system trusted at the B2 level of the Trusted Computer Systems Evaluation Criteria or higher.

It is also very difficult to determine how much of the database management system needs to be trusted. Any portion of the system that has the potential to modify the actual data or audit logs could be considered part of this security kernel. In theory, database management systems support separable functions, however, in practice, there is some debate as to the number of commercial database management systems constructed totally out of modular sections. Many of today's database management systems are highly integrated and the modules which support these individual functions are not always distinct or interchangeable. Therefore, since it is difficult to distinguish between the functional modules, different database management system designs require different portions of the database management system to be trusted. This leads to the conclusion that those portions of a database management system that must be trusted are determined by three primary factors: 1) the design of the database management system, 2) the design of the security mechanism within the database management system, and 3) the interrelation between the operating system security mechanisms and the database management system's security mechanisms and policy.

Beyond these conclusions, there have been arguments that the only way to ensure that the data is protected, especially in the multilevel environment, is to include the database as a protected object that is isolated from the control of the standard system file structure to some degree. This eliminates the possibility of access through standard system file commands and forces the user to access the database through the database management system. There still must be some consideration of applications provided with the database management system (spreadsheets, report generators, etc.) that manipulate data after it has been retrieved but before it is delivered to the user in the requested format, and easy query language interfaces that convert user generated English statements into Structured Query Language expressions that can be executed by the query processor.

The incorporation of security features into a commercial database management system is not an easy thing to do. Beyond finding a way to secure or control the operating system interfaces, a large portion of the database management system itself might require revision or replacement to eliminate or narrow potential covert channels. The dependencies between the operating system and the database management system are very complex. In the distributed database environment, they become even more difficult because network security must also be considered. With the backend database machine, the question of how much confidence exists in the host request mechanism must be addressed.

From the four alternative exploratory architectures discussed, perhaps the architecture with the highest potential for the greatest security is the fourth alternative, the customized combined operating system/database management system kernel approach. This approach would address the efficiency concerns inherent with security kernels as well as the performance considerations for database management systems. The security policies of the database management system and the operating system could be more easily reconciled Because they would be developed concurrently and with a greater degree of confidence that the end product was secure.

A trusted database management system will not be built overnight. Rather, it must be carefully constructed to afford the maximum protection possible to the data, a sufficient audit trail, and a thorough recovery process to eliminate data inconsistencies that may result from crash. All of these features must exist, and performance penalties must be minimized. It may not be possible to incorporate all of these features in a near-term solution. However, worked examples of the various security techniques must be created now to be incorporated into the secure data management systems of tomorrow. To do otherwise would result in the ultimate secure system -- one so secure that nobody could afford the price of its use.

Bibliography

1. Department of Defense Trusted Computer System Evaluation Criteria, CSC-STD-001-83, 15 August 1983, DoD Computer Security Center, Ft. Meade, MD.

2. Department of Defense Evaluated Products List (EPL) for Trusted Computer Systems, 2 April 1985, DoD Computer Security Center, Ft. Meade, MD.

3. Henning, Ronda R., "Multilevel Application Development", Proceedings of Eighth National Computer Security Conference, NBS, 1985.

4. Lampson, B.W., "A Note on the Confinement Problem", CACM, October 1973.

5. Hsaio, David K., "Data Base Computers", Advances in Computers, Vol, 19, 1980.

6. Date, C.J., An Introduction to Database Systems, Volume 1, Fourth Edition, Addison Wesley, 1986.

7. Stonebraker, Michael, Editor, The Ingres Papers, Anatomy of a Relational Database System, Addison Wesley, 1986.

8. Air Force Studies Board, Committee on Multilevel Data Management Security, "Multilevel Data Management Security", National Academy Press, 1983, Washington, DC.

9. Neumann, P.G, et al, A Provable Secure Operating System: The System, Its Applications, and Proofs, CSL-116, 7 May 1980.

GUIDELINES AND STANDARDS

Carole S. Jordan

National Computer Security Center
9800 Savage Road
Ft. George G. Meade, MD 20755-6000
(301) 859-4452

INTRODUCTION

This paper describes four guidelines and standards that have been or are being developed in the Standards Division of the National Computer Security Center. These documents are: DoD 5200.28-STD, DoD Trusted Computer Systems Evaluation Criteria, Draft DoD Directive 5200.28, Security Requirements for Automated Information Systems (AIS), Trusted Network Guideline, and A Guideline on Office Automation Security.

DOD 5200.28-STD

The Department of Defense Trusted Computer System Evaluation Criteria, CSC-STD-001-83, was signed as a DoD standard by Mr. Latham, the Assistant Secretary of Defense for Command, Control, Communications and Intelligence (ASD(C3I)), in December, 1985. The standard is DoD 5200.28-STD, entitled Department of Defense Trusted Computer System Evaluation Criteria.

The standard is nearly a duplicate of CSC-STD-001-83. During coordination within the DoD, however, some changes were agreed upon between ASD(C3I), the NCSC, and the DoD components. A document was created that contains a summary of the changes that were made. This document is being distributed along with the standard. The standard has been printed (It, too, has an orange cover.) and copies are available from the NCSC.

DRAFT DOD DIRECTIVE 5200.28

Background. The Secretary of Defense tasked the Assistant Secretary of Defense for Command, Control, Communications and Intelligence, ASD(C3I), in collaboration with the NCSC to revise the directive. To accomplish the task, a task force chaired by NCSC was formed of DoD representatives who have computer security expertise, are familiar with current DoD policy, and are aware of their own components' security needs. A draft directive was produced by the task force and sent to ASD(C3I) for their review and coordination among the DoD components.

Overview of the Draft Directive. The SECDEF had three objectives to be accomplished in the revised directive. The first objective was to ensure that the directive applies to all computer-driven information systems. The second objective was to add policy guidance for including computer security requirements in AIS procurements. The third objective was to incorporate the use of computer security standards and guidelines. These objectives were accomplished by the task force during the rewrite.

The third objective was accomplished by introducing the DoD 5200.28-STD in the draft and by requiring its use in the selection of security features that will meet the requirements stated in the directive. Without going into detail, the following is a brief description of how the draft directive incorporates the DoD standard:

All AISs that handle classified or sensitive information must have security safeguards that are adequate to meet a set of minimum requirements specified in the draft directive. These minimum requirements are similar to those listed in the original DoD directive, but they have been reworded and updated. The minimum requirements include such things as individual accountability, audit trail, access control, physical controls, and appropriate marking of output products.

For those AISs that will operate in the dedicated security mode, the set of minimum requirements may be met by automated or manual means, and there are no further requirements to be met.

For those AISs that will operate in the system high or multilevel or partitioned security modes, where there is increasing risk involved in the protection of the information being handled by the AISs, there is further guidance in the draft directive that must be followed in order to determine the additional security safeguards that are necessary.

The guidance in the draft is comprised of a series of steps that must be taken to determine the requirements that must be met for a particular AIS. The first step is to determine the security mode of operation from among the modes that I listed above. The second step is to determine the minimum user clearance, or, more precisely stated, the maximum clearance of the least cleared user. The third step is to determine the maximum sensitivity of the information handled by the AIS. The information derived in steps two and three are assigned values, and in step four the values are used to produce a risk index. In step five the risk index is mapped to a particular evaluation class in the DoD standard. As an example, a risk index of 2 maps directly to class B2 in the DoD standard, indicating that the AIS must meet B2 requirements. The information in these five steps is the same information as that found in the publication entitled Computer Security Requirements -- Guidance for Applying the DoD Trusted Computer System Evaluation Criteria in Specific Environments, that was produced by the Standards Division in June of 1985.

Several other changes to, or departures from, the original directive were made by the task force. For instance, the current directive applies to the protection of classified information, whereas the draft applies to the protection of classified and unclassified but sensitive information.

The Designated Approval Authority (DAA) is introduced in the draft. Most DoD components have already defined and incorporated the term in their own implementing regulations, so updating the directive on this issue made it current with the implementing regulations of other DoD components.

The responsibilities of the System Security Officer (SSO) are expanded in the draft directive. In the current directive, the SSO is not appointed until an AIS is operational. In the draft, it is required that someone be appointed the SSO early in the life cycle of a new AIS to ensure that security is considered during the design and development stages.

As in the case of the DAA, there were several other issues in the draft directive that were updated to bring the draft in line with DoD implementing regulations.

Status. The draft directive is currently being coordinated among the DoD components for their concurrences and comments.

TRUSTED NETWORK GUIDELINE

Background. The Standards Division of the NCSC began a project in late 1983 to draft what were then known as Trusted Network Evaluation Criteria. An invitational workshop was held in New Orleans in March, 1985, to obtain input from the DoD, from private industry such as vendors and users of computer networks, and from the academic community. Using material produced in the workshop, a draft Trusted Network Evaluation Criteria was developed and published in July, 1985. The draft, informally known as the Brown Book, was distributed to several hundred reviewers for their comments. Comments received from the reviewers were extremely disparate, and it was concluded that the Brown Book could not be modified to satisfy the diverse viewpoints of the reviewers. The Brown Book was scrapped and a different approach was taken. A working group was formed to interpret the DoD Trusted Computer System Evaluation Criteria (TCSEC) for computer networks and prepare a draft guideline.

Overview of the Proposed Guideline. The Trusted Network Guideline (TNG) will apply only to those networks that can be thought of as having one trusted network base (TNB). There are other types of networks, and there are internets that are in some sense also networks. These networks do not support a single TNB, and, therefore, it may not be meaningful to assign a rating to them in the way that we could assign a rating to a network with a single TNB.

The TNG will apply only to those networks that provide all users with an interface that is at the same level of trust. Other networks should be connected using what will be called "interconnection rules," which will be provided either as an appendix to the TNG or as a separate document.

Tentatively, the TNG will be comprised of criteria (from the Orange Book), interpretations of the criteria for networks, and rationale for the interpretations. New requirements will be added to address integrity and denial of service issues. These issues are significantly more important for networks than they are for stand-alone systems.

Status. The draft is being developed and, once the working group is satisfied with it, will be distributed to a larger group of reviewers. The draft will then be revised as necessary and published and reviewed by as wide a community as reviewed the Brown Book. It is our goal to have a comprehensive draft document published by the end of this year.

GUIDELINE ON OFFICE AUTOMATION

Background. The Standards Division of the NCSC was tasked by the Standards and Guidelines Working Group of the Subcommittee on Automated Information System Security (SAISS) with developing a guideline on Office Automation Security. The goal of this effort was to produce a document that would provide guidance for all OA systems in the Federal Government that are used to process classified or other sensitive information. The document that has resulted from this effort is entitled A Guideline on Office Automation Security.

Overview of the Guideline. This guideline is intended to provide guidance to users, security officers, procurement officers, and others having responsibility for the security of an office automation system at some time during its life-cycle.

The guideline is divided into four parts. Part I is an introduction and overview. It contains the introduction, purpose and scope of the guideline, as well as a high-level overview of why the office automation security problem is different from other computer security problems.

Part II of the document provides security guidance for users of OA systems. The class of users includes secretaries, managers, technical and non-technical users, and others. Therefore, this part of the document has been carefully written to be understandable by all who need the guidance it gives.

Part II contains chapters on the security responsibilities of OA system users, operational security guidance for stand-alone OA systems, and operational security guidance for connected OA systems.

Part III of the guideline provides guidance for OA System Security Officers. There is a chapter that outlines some of the security responsibilities of the SSOs, and a chapter that discusses various threats, vulnerabilities and controls that they should be aware of.

Part IV of the guideline gives guidance to others. There is a chapter outlining some of the security responsibilities of the organization that owns or is otherwise responsible for the system. There is a chapter that gives guidance to procurement officers concerning important points to consider when it is time to acquire an OA system. There is also a chapter on the secure disposal of both the OA system and the magnetic storage media that is used in it.

In addition, there is an appendix that provides a guideline on sensitivity marking for the OA system and its storage media. This appendix suggests a scheme for the physical labeling of equipment to help prevent accidental compromise of classified or other sensitive information.

Status. Drafts of the guideline have been reviewed by members of the Working Group, as well as by members and observers of the SAISS. It will be voted on by both the SAISS and the Subcommittee on Telecommunications Security. If approved, then the NTISSC will decide whether or not to release the guideline as an Advisory Memorandum. The guideline should be available for public release in the near future.

PANEL

ON

DATABASE MANAGEMENT SYSTEM SECURITY REQUIREMENTS

We rely on databases in the defense of this country, to support our financial and legal systems, in our medical and educational systems, and even to receive our paychecks. Very serious consequences could result from the penetration and/or alteration of these systems.

According to an Ohio University Study in the September 9, 1985 issue of Computer & Software News, seventy percent of the top videotex and database service firm executives considered unauthorized access to be a significant problem. Ten percent reported that tampering occurs on a weekly or more frequent basis. Another ten percent reported that tampering incidents occur monthly. Thirty-two percent cited other intervals of frequency.

Since the 1982 Summer Study on Multilevel Data Management Security, several operating system products have appeared on the Evaluated Products List and many more candidates are being evaluated. Today, however, an unsecured database management system, placed on a trusted operating system, produces an overall system where the data is poorly protected.

A primary research emphasis of the National Computer Security Center has been the development of secure operating systems. With the first of these products developed, we now turn our attention to an even more difficult area: database security. The primary guidance that the Center and vendors have had on secure data management has come from the Summer Study report, which details near- and long-range goals and objectives for secure data management research. Additional input has been obtained from the Center's technical review group and from the recent workshop on database security.

This panel will review the validity of the findings of the Summer Study, open a new forum for discussion on what the user community sees as their current and future requirements for secure data management, and present a brief synopsis of database security research in progress. It would serve as a kickoff to a general data call planned by the Center's Secure Database Research and Development Branch to determine its direction in database security research.

Panel Chair:

Dr. John R. Campbell, National Computer Security Center

Panel Members:

Dorothy E. Denning, SRI International

Kenneth Eggers, MITRE

Roger Schell, Gemini Computers, Inc.

Charles J. Testa, Infosystems Technology, Inc.

PANEL DISCUSSION

NCSC AND VERIFICATION

Formal Specification and Verification are important technologies in the production of secure computer systems. As development of A1 (and beyond A1) systems increase, greater demands will be placed on the automated verification tools and the developers and maintainers who support their use.

The National Computer Security Center (NCSC) has made a commitment to support formal verification. What does such a commitment mean, and what is the NCSC doing to fulfill that commitment?

The primary focus of this panel discussion is to identify the role and commitment of the Center concerning the formal specification and verification technology. The discussion will include the following topics:

a. The need for verification (introduction).

b. The Product Evaluator's Verification Working Group. This working group was created to help evaluators with verification issues concerning A1 or beyond-A1 evaluations. A description of the working group's charter and progress are presented.

c. Endorsed tools. Questions such as "What does endorsed mean?" and "How can a verification system be added or deleted from the endorsed tools list (ETL)?" are discussed.

d. Future endeavors (1 year).

e. Milestones. The Center has been in the forefront of verification activities. Such activities are highlighted.

f. Future technology. Thoughts of where verification technology will be in 5 years.

The panelists are representatives from all offices within the NCSC and two recognized verification experts outside of the NCSC. The format for this panel session is to have each key panelist talk for approximately 10 minutes, after which the panel is open to questions from the floor.

PANEL DISCUSSION

Using the Criteria in Acquisitions

Incorporating trusted system computer security related requirements into acquisition programs is a difficult task faced by managers procuring trusted systems. The evolution of the Department of Defense Trusted Computer Systems Evaluation Criteria from a guideline (CSC-STD-001) to a Department of Defense standard (5200.28-STD) will certainly add to the number of acquisitions requiring trusted systems. Thus, this panel is geared to provide the audience with information about real world trusted system acquisitions and how to integrate security, acquisition, and program requirements. The panelist are key players involved in program acquisitions ranging from class B1 - A1. The topics include:

- Computer Security Acquisition Management
- Procurement Guidelines for Multi-level Systems
- Applying the Procurement Guidelines at Class B1
- Interservice/Agency Automated Message Processing Exchange (I-S/A AMPE) Experience
- The BLACKER Program and the Criteria
- The FORSCOM SECURITY MONITOR (FSM) Lessons Learned

The panelist are Mrs. Suzanne O'Connor, Standards and Products Office of the National Computer Security Center (NCSC); Miss. Leslee O'Dell, Special Projects Office of the NCSC; Mr. Gregory Elkmann, Automated Information System Evaluation Office of NSA; Mr. H. O. Lubbes, Space and Naval Warfare Systems Command; and Captain William Collier, Automated Information System Evaluation Office of NSA. The panel chairman is Major Donald Baker, Technical Support Office of the NCSC.

AN ECONOMICALLY FEASIBLE APPROACH TO CONTINGENCY PLANNING.

Robert H. Courtney, Jr.
Robert Courtney, Inc.
Box 836
Port Ewen, New York 12466
914-338-2525

INTRODUCTION

What is a Contingency Plan?

A contingency plan describes the appropriate response to any situation which jeopardizes the safety of data or of data processing and communications facilities to a degree that threatens meaningful harm to the organizations supported by those data and facilities. A contingency plan is not a book; it is an action plan.

The threatening situation need not be a disaster which causes extensive physical damage. The disruption may cause no damage at all to the physical facility as is often the case with a chemical spill which, by forcing the evacuation of personnel, stops data processing activities. In fact, the economic feasibility of a contingency plan may well lie in its ability to contain small problems at small cost as well as providing the ability to fare through the total loss of a physical facility.

The threats to be anticipated in devising the contingency plan need only be sufficiently great in both the magnitudes of the potential losses and in their probability of occurrence to justify the preparation of plans to avoid those losses if a course of action which costs significantly less than the anticipated loss can be devised.

It is regrettable that the term "disaster recovery plan" has become, for many, synonymous with "contingency plan". It seems somewhat more rational to consider the contingency plan to be a disaster avoidance plan rather than a way of recovering from a disaster. Most of our data processing disasters become such only because we are not prepared to cope with what might have been only an inconvenience if we had prepared properly.

Who Needs Them?

Any organization which is susceptible to significant harm if it loses its data or the facilities associated with their use needs a plan with which to respond to reasonably anticipatable disruptions to normal data processing operations. These can include labor problems as well as earthquakes, leaking roofs as well as floods, gross mistakes by loyal employees and bombs by terrorists, area-wide losses of power and vital communications lines cut by back-hoes.

The losses which mount as a consequence of system outages vary widely with the nature of supported organizations. Some major organizations will not be seriously hurt with downtimes as long as a week. Others will suffer meaningful losses, amounting to as much as two-thirds gross revenue, starting within minutes of loss of system support.

Who Has Them?

Truly workable, fully tested, economically feasible contingency plans are in place for only a small percentage of the data processing mainframe installations. There are no untested but workable contingency plans. Such tests always reveal deficiencies to be fixed.

It is our belief, based upon many discussions of the subject with DP management and others, that the principal reason for the absence of good contingency plans, at least in the private sector, but to a lesser degree in the public sector because of the many complicating factors there, is the continuing belief by much of the DP management that workable plans are far too costly or are, in reality, infeasible. Other important and more urgent issues do divert management attention from contingency planning and other security related considerations as well, but the principal barrier seems to be lack of confidence that a truly workable plan can be configured. Until more DP directors are better informed about the economic feasibility and workability of contingency plans, this situation will not change.

Our goal here is to describe an approach to contingency planning which is clearly workable in many, but certainly not all, organizations.

THE MAJOR COMPONENTS

We are addressing here contingency plans for data processing, including communications, data acquisition, storage, and presentation. Of no less importance, but not within the scope of this paper, are the contingency plans for the critical dependencies which are not DP related. Preservation of the ability to take orders and bill customers may lack importance if there is no means of making shippable product.

The essential components of a complete contingency plan are these:

1. Emergency Response Plan.- A plan to respond promptly and well to a potential disruption so as to limit the damage is highly desirable. Fire extinguishers are almost worthless if no one knows how to use them. In this category, then, are the things which should be done as soon as there is an awareness of a potential problem which might result in the invocation of the contingency plan.

2. Back-Up Plan.- The back-up plan provides the ability to conduct, by alternate means, the critical data processing workload. The critical workload is that portion of the workload which will generate serious loss if disrupted for a period exceeding two weeks. See our comments later here on the selection of the two week period.

3. Recovery Plan.- The Recovery Plan guides the return to full and normal data processing capability.

All three plans must be considered because all are important, but the first two, the emergency response and back-up plans, are the most difficult to put in place and, usually, are the most urgently needed. There is rarely any significant overlap of the three categories. This paper is oriented primarily toward the provision of economically feasible back-up capabilities.

THE ALTERNATIVES

Several different approaches to the provision of back-up capability can be considered. They are not all equally workable. These should be considered only under some quite exceptional circumstances. The principal alternatives, then, are these:

Doing Nothing.

There are a few organizations quite dependent upon computer-based systems which will suffer but little loss if they are without that data processing support for two weeks or so. Such loss as they might encounter if they cannot run their work will be quite small in comparison with the cost of a back-up capability. Those charged with contingency planning should consider the highly desirable possibility that their respective organizations may be in this category.

It is not wholly uncommon to encounter the absence of need for back-up in headquarters operations where computers are used primarily for planning and higher level awareness and control purposes and where accounting, payroll, order entry, inventory management and other such time-dependent tasks are provided for the enterprise by DP shops in the operating divisions. Note that these other DP shops do need back-up capabilities.

Mutual Aid Agreements.

External. Arrangements made with other, unaffiliated organizations to provide back-up data processing support by deferring some of the supporting company's less critical work can work under some circumstances. It is usually fairly easy to arrive at some informal agreement of this type with other organizations. It is more difficult to establish formal written agreements which are workable. It is usually quite difficult to establish such mutual aid agreements involving adequate, periodic tests of that back-up. In general, and as we stated rather forcefully above, untested back-up plans do not work.

These arrangements increase in workability under the following circumstances:

1. When there are unused shifts available at the back-up facility so that less work, if any, is displaced in the supporting company.
2. When the work to be backed up is primarily vanilla batch or with limited use of in-dial ports only.
3. When the CPU's are relatively small.
4. When the need for back-up is such that delays of a few days will not be very costly.
5. When the mutually supportive organizations are in the same industry areas; e.g., commercial banking. That two companies are possible competitors is often an impediment, but usually not so great a problem as a complete lack of appreciation of what the other is trying to do as when they are in different businesses.
6. When the two companies are of roughly the same size.

None of the above factors are without notable exceptions, but they should provide some useful guidance in considering a mutual aid agreement with another organization.

The most useful observation we can make here about mutual aid agreements between wholly unaffiliated organizations is that they very rarely work when they are needed.

Internal. The workability of mutual aid agreements between groups with some organizational affinity is dependent upon many factors. The more prominent of those factors are these:

1. The strength of the stated desire of the common management that the respective organizations arrange such back-up support.
2. The quality and the degree of realism reflected in the back-up plan.
3. The conduct of wholly realistic tests of the back-up capability.
4. The similarity of the mutually supportive systems.
5. The simplicity of the required communications support.
6. The physical proximity of the two sites - provided that they are not so close as to be affected by the same source of disruption.

7. The availability of time to correct deficiencies in the back-up support after disruption and before losses mount intolerably.

Many other things can be listed, but these deserve careful consideration before this option is elected.

The greatest single factor in the workability of this arrangement is the ability and willingness of the common management to require the provision of fully tested back-up. Other factors are very important, but this one is usually key.

Open Hot Site.

An open hot site is a data processing facility operated for profit by making available to otherwise unaffiliated companies a site on which they can conduct their data processing after loss of the use of their own facility. Those are characterized by the facilities of COMDISCO and SUNGARD.

Monthly subscription rates are paid to preserve the ability to test the back-up plans and, when necessary and on payment of additional fees, to declare an emergency and move the ‹ itical workload onto this alternate facility.

This arrangement is indeed quite workable for a number of companies, but it is far from a universal solution. It can be a partial solution for some banks, for example, but it will not solve the problem of data capture, including the proof operations, so essential to curtailing losses through disruption to demand deposit operations.

An analysis of the economic feasibility of the open hot site as back-up for any specific facility must include careful and quantitative consideration of the speed with which key data processing functions must be restored. The feasibility of this approach clearly increases with the length of time available to move people and data, to fix unanticipated problems, and to adapt the hot site communications facilities to the peculiar needs of the using organization. The cost of repeated tests at a geographically remote location must also be considered in evaluating this alternative.

Although it may be argued that such should not be the case, we have seen too many instances in which recovery at the hot site has been deferred for an inordinately long period while attempts are made to recover at the primary location to avoid paying the fees for declaring an emergency or because there is fear that the contingency plan may not actually work. Further, if there is some reasonable possibility of a prompt recovery at the primary site, prudent management will be reluctant to send the best people available to the hot site, as will be needed to establish operations in a different location, when it is clear that operations cannot be re-established at home without those best people. This is a difficult dilemma for the DP Director to resolve when he is faced with the plethora of problems normally encountered when a busy facility is suddenly and seriously disrupted.

Closed Hot Site.

A closed hot site is a facility which is owned by a consortium or which was otherwise constructed for a specific set of companies to satisfy some less than highly general need for back-up capability. Such a facility might provide proof machines and operators and check sorters for a group of banks. It might provide unusually rapid availability of back-up for organizations which encounter serious losses beginning with the first minute of facility outage.

These facilities are rare primarily because of the heavy requirements for a peculiar combination of entrepreneurial spirit, salesmanship, technical strength, and quite substantial investment (by the participating companies) needed to get them to an operational state. They can be a highly satisfactory way of satisfying the back-up needs of enterprises which cannot afford to be down for even very short periods, but the costs are significantly greater than those seen with open hot sites. These higher costs are justified only when they are fully displaced by sum of the losses avoided by this approach and the continuing availability of the facility to the participating companies for rehearsal of back-up plans and for application development and test.

This approach is definitely not the way of the future for very many organizations. It is very good for those who need it, but it will not be economically feasible for very many others. In some of those organizations for which it would be the correct approach, the DP management will not find it acceptable to ask the corporate management for the necessary funds to participate in such a consortium.

Split Sites.

Later in this paper we discuss the determination of the size of the truly critical workload in a DP mainframe facility.
For our immediate purposes here, it is sufficient to say that it is very rare to encounter a conventional data processing facility supporting a multiplicity of applications on a mainframe where the truly critical workload approaches 50% of the total. Our definition of critical workload is that portion of the workload which, if discontinued for two weeks, would result in serious loss, not just inconvenience, to the enterprise. Most commonly, if a reasonably objective evaluation is made of critical workload, it will be less than 20% of the total.

We have found it generally quite feasible to return to a reasonable semblance of normal operations within two weeks of even a major facility loss. For this reason we use the two-week period in our definition of critical workload.

If the critical workload on a facility is significantly less than 50% of the total, then it is possible to consider splitting an existing site into two physically separate parts either of which is large enough to carry the critical workload. At least theoretically, this does not require any increase in data processing capacity. In actual practice, that is not quite correct.
However, with a split site, if either is lost, the other can carry the critical workload after shedding that portion of the non-critical workload it was carrying before loss of that other facility.

It is our contention that, while other approaches to back-up are sometimes workable, the most broadly applicable, economically feasible approach to backing up critical workloads on mainframes is the split-site arrangement.

Standby Facilities.

We noted above that it is rarely possible to provide economic justification for a whole standby facility which does nothing until the primary one is lost. It is possible to compose a scenario in which the consequences of losing a facility are so dire as to provide such justification. This is most commonly true with smaller, dedicated machines such as those driving automated warehouses.

In the whole population of computers, there are enough situations where dedication of otherwise unused back-up facilities are justified that that category cannot be excluded from any reasonably comprehensive list of alternative approaches.

Data Servicers.

Many organizations would be well-served in any attempt to reduce or eliminate the critical workload to consider taking all or a portion of it, depending on its nature, to a data servicer such as ADP or McAuto. They not only might substantially reduce the cost of operations such as payroll, they can also have advantage of the extensive facilities of the larger organizations in that business to assure a high probability of the continued support of those delegated functions. In general, however, the time to place the work with the data servicers is before and not after disruption to your facilities.

A complete discussion of the several reasons for taking payroll and some other common business functions to outside specialists is somewhat beyond the scope of this paper, but the reader should give it appropriate consideration.

Combinations of Alternatives.

It is readily apparent that some combination of the different alternatives may best suit the needs of very large organizations and even a few of the very small ones. For example, a bank may well consider taking its payroll to ADP, using an open hot site for its mainframe back-up, and joining a consortium for proof and sorting operations. Many other equally plausible examples can be cited.

THE CRITICAL WORKLOAD.

What is the Critical Workload?

Earlier here we said that the critical workload is that portion of the total workload which would cause serious loss if it could not be conducted for periods of up to two weeks. The two weeks is fairly arbitrary but, in reality, most companies do manage substantial recovery of their data processing operations in that time even when there has been catastrophic loss of a major facility. If the two-week interval seems inappropriate to any particular environment, it is quite reasonable to pick some longer or shorter period, although much shorter might be quite risky.

Another perspective on the problem might suggest that the critical workload is that portion of the total which, if it is interrupted, would generate losses great enough to provide economic justification of a back-up capability which would obviate such losses. This view of things is not correct because is suggests an assessment of criticality based on cost of back-up. The desirability of avoiding loss does not change with the feasibility of avoiding it.

All of the potential losses which would result from an outage should be compiled, not just those which can be obviated by some current notion of the nature of an appropriate back-up capability. Only when those are available will it be possible to configure a back-up plan which is sufficiently detailed to be workable. When these potentially avoidable losses have been compiled, then we can evaluate the various approaches available to us for providing a back-up capability and select the combination which displaces the greatest potential loss for the least cost.

EAL = (Cost)(Probability of Occurrence).

The Expected Annual Loss (EAL) which is used to justify backing up a data processing function or not should be evaluated in terms of not just the dollar consequences of an undesirable event but also the probability that it will happen. It is not reasonable to base a contingency plan on an assessment of consequences alone; consideration must also be given the probability (or frequency) of encountering the interruption. It is clear that the anticipated loss must be the result of consideration of both the damage done and the chance of encountering the problem.

When doing a risk assessment for contingency planning purposes, it is usually far easier to assess the consequences of a disruption than it is to judge the probability of encountering the problem. Fortunately, quite gross estimates of which we can be reasonably certain are usually good enough. No attempt should be made to refine data beyond the point where the improved precision does not make a difference in what we do. Only when we realize that the determinant for a course of action lies in the area of uncertainty between the upper and lower bounds of the value we assign a parameter are we justified in expending the effort to further refine those data. Distinctions without differences are useless; and for there to be a difference, a change has to make a difference.

We have found fairly consistently that we are rarely hampered by difficulty in estimating probabilities of occurrence. Far more often than not, when we take the probability to a level so low that we are quite comfortable with it, the consequences are sufficiently large to provide adequate justification of corrective measures. This should not be too surprising because the things with the most dire consequences usually happen with the lowest frequencies - otherwise the world would not be habitable. On the other hand, small problems often happen with frequencies so high that they rival or exceed the EAL of the catastrophes.

If, for a particular problem, we cannot arrive at an estimate of probability (or consequences) in which we have adequate confidence, it is often best to simply defer further consideration until later. Quite frequently, the appropriate corrective action will be cost-justified by some other problem which is more readily assessable. If that does not happen, then we must do the additional work required to further refine our data.

In conducting a risk assessment, a small amount of common sense far outweighs complex methodologies. We once encountered such blind obeisance to a flawed risk assessment methodology that, in a prioritized list of critical data processing tasks for a major manufacturing company, paying suggestion awards was ranked first and higher than accepting orders, shipping product, invoicing, receiving payment, and getting out the payroll.

Who Determines the Critical Workload?

Involvement of Functional Area Managers. Our experience has been that determination of the critical workload by the information systems personnel working alone and not in close cooperation with the managements of the respective functional areas does not work. The DP people almost never have the depth of understanding of the need of the enterprise for the proper functioning of each of the essential components to the extent that they can offer a quantitative evaluation of the cost of their interruption. All too often, they don't know that they don't know and, as a consequence, make gross and seriously erroneous estimates of the tolerance of the organization to specific problems. Their assessment of the importance or criticality of particular functions as often reflects the strengths of the personalities of the persons from that area with whom they have been working as it does the real situation.

Importance of Policy Statement. We have almost always found it more difficult to achieve the involvement of the functional area managers in any planning for computer-related security, including contingency planning, in the absence of a strong policy statement issued by the chief executive officer. The policy statement should make specific assignment to functional managers of direct responsibility for the safety of data and the means of processing them. The DP management should have custodial responsibility for data and an obligation to extend to the data and the processing means such safeguards as may be required to contain the concerns of the managements of the directly responsible functional areas. The workability of this arrangement is greatly improved if the cost of security as defined by the functional managers is charged back to them. This provides an incentive for them to balance their concern for data security, for which they are held accountable by a proper policy statement, against the cost of providing it so as to make certain that no more is spent protecting data than it would cost to leave it unprotected.

Questionnaires versus Interviews. We know of no paper survey of computer security matters with other than extremely limited scope, such as which access control method has been procured, which has yielded data which are both accurate and useful. It is far easier to acquire, by proper questionnaire design, data which seem useful than it is data which are accurate. For example, there was in the Department of Defense a contingency planning questionnaire which asked, "Is your system is subject to acts of God?" We never saw that answered by other than a "No".

A major problem with paper surveys in the computer security area is that the amount of explanatory text which must accompany the questions so badly burdens the task of preparing the surveys and answering them that either the writing or the reading (or both) of that material is too often neglected.

We have found eyeball-to-eyeball interviews with key managers of functional areas by far the most satisfactory and least time-consuming approach for everyone concerned. The skilled interviewer should be accompanied by a person from the DP area - preferably, the person who will be charged with maintaining the contingency plan after its preparation - so as to provide learning for him in the conduct of those interviews. With such an arrangement, it is usually relatively easy to reach agreement between the contingency planners and the functional managers as to the direct and, very importantly, indirect costs of losing data or the processing means as a function of the duration of such loss. As we noted above, the frequency or probability of occurrence may be a little more difficult, but it is not an overwhelming burden.

THE SPLIT-SITE APPROACH

The most commonly encountered major problems in achieving and retaining a truly workable back-up plan are these:

1. Identifying the critical workload. Not only must it be initially identified, it must be continually assessed as new applications evolve and as the organization's priorities change.

2. Assuring suitably prompt availability of adequate computer(s) when they are needed.

3. Establishment of enough of the normally required communications network to provide adequate limp-along capability.

4. Conducting realistic tests of the contingency plan in the face of opposition to the cost of the tests, to the disruption to non-critical workload and to the potential for disruption to the critical workload.

5. Assuring availability of data. With the rapidly growing size of some data bases, this problem is becoming increasingly severe if only because of the time and costs required to unload and load the amount of data required to support the critical workload. Planning and assuring the availability of back-up data has become an important and fairly costly part of good systems management. It is wholly essential to workable contingency plans.

6. Assuring the availability of the skill levels required to respond promptly to a need to back-up the critical tasks and to phase off-line gracefully the non-critical tasks.

7. Maintenance of management support for contingency planning.

8. Preservation of an awareness on the part of planners and developers that ease and cost of back-up and recovery should be weighed along with all of the other operable factors when planning new applications and the refurbishing of old ones.

The list above is not necessarily in priority sequence. The relative importance of these things will vary with the organization.

Now, given a wide variety of candidate approaches to back-up, most of which were listed earlier (with notable and generally unworkable exceptions, such as dedicated and unused floor space with no DP hardware) and given these more common difficulties, we must pick an approach that promises the greatest potential workability. More often than not, it is the split site.

While the split site is most often the most workable approach to back-up, it is not necessarily the approach chosen even when it is the most appropriate. Too often there is an unwillingness to solicit management support for any approach which will have significant cost, even when it is cost-justified. If the cost will require a diversion of resource which would otherwise be available to increase data processing services or if the implementation of a good back-up plan would require recognition by the senior management of the high jeopardy with which they have been living for some time while assuming risks about which they had not been told, the DP management may opt for a less workable, basically cosmetic approach to back-up which avoids rousing the ire of that senior management. For our purposes here, we will assume that there is a sincere, politically unfettered desire to implement the most cost-effective plan.

The split-site approach is not always the best approach, but, more often than not, it provides the most cost-effective, workable one with the least encumbrance by the several negative factors listed above. We will now attempt to support that assertion.

We stated above that it is very rare for the critical workload to exceed 50% of the total workload and, more commonly, it is in the order of 20% of the total. Even during first shift on systems with heavy interactive, real-time loads, well more than 50% of the work is usually divertable to the non-critical category. When such is not the case, the most burdensome applications should be examined to see whether some of the work done under them should not have been relegated to batch and is, instead, being done unnecessarily in the real time environment.

If less than half of the total workload is critical, then it is clear that, at least conceptually, we can convert a single facility into two without increasing the total capability and have either of the two parts be large enough to carry the critical workload. Under these circumstances, we would not need to involve the facilities of other organizations to have a back-up capability. It is clear that systems do not cut cleanly into two parts of precisely the relative sizes we might want, but that is not a major problem.

Split-Site Costs.

It is clear that one cannot divide an existing facility into two parts easily or without added cost. If it is planned carefully, however, it can be done at costs sufficiently low as to make it an attractive proposition for most organizations. The smaller the critical workload, the smaller the second facility must be and, normally, the lower the cost of establishing and operating that site.

The economies of scale dictate it to be less expensive to carry a workload in one location rather than two or more. This is not a commentary on the desirability of distributed processing or putting DP under the direct control of the functional areas supported. A given DP workload is normally less costly if it is done all at one place. Because, in this case, there is a reason for splitting the workload between locations, we want to do it in such a way as to minimize that cost.

It is reasonably obvious that the smaller the second site, the less the increase in the operating cost of the two sites over the cost of the initial single site. Thus, the second site should be as small as possible and still carry the critical workload and, of great importance, be a fully viable facility for carrying whatever normal workload is appropriate for placement there. We have found that the second site increases the operating costs of that portion of the work brought to the second site by about 20%. Thus, if 20% of the workload is moved to a new site which has a capability of about 20% of the initial facility, the increase in costs incurred by operating at the two sites instead of one will be roughly (0.20 X 0.20) or 4%. The 20% figure is useful only for initial guidance and should be confirmed by hard estimates of the costs in the specific operating environments under consideration. Many factors may influence its actual value.

If the 20% increase can be confirmed for a specific environment, then it is clear that the split-site provides a highly desirable back-up option if there are no other significant barriers to that approach.

Dividing the Workload for Split Sites.

There is probably no need to note here that, when split sites are planned, as much as possible of the critical workload should be placed in the facility which is least likely to be disrupted for any reason. This is not always practicable, but, where it is, it should be done.

Many organizations already have a split between processors handling the normal workload and those supporting development and test, although these processors are often in the same physical area and are, therefore, jeopardized by the same infrastructure disruptions. It is not uncommon to find that the test and development capability is large enough to carry the critical workload provided only that appropriate access to essential communications and DASD can be provided.

Because test and development is almost always a prime candidate for suspension when normal processing is disrupted and back-up of critical applications is required, running them in the site most likely to be used for back-up often affords an ease of transition to the critical work when that is needed.

Putting test and development at the most secure site might, because it would not be needed normally, preclude the availability of continuing attachment of that site to those communications facilities needed to support the critical applications. Even though there might be an intent to preserve the ability to provide the communications necessary to the back-up capability at that site, it is terribly easy for that ability to atrophy unnoticed and not be available when needed. Care must be taken to avoid that problem.

If the normal workload at the second site requires availability to the communications facilities which support the critical applications, then no hardware or extensive logical shifts need be made to run those backed-up applications there. It is quite fortunate when such an arrangement is feasible.

If all facilities on which the critical applications might be run when back-up is needed are on the same SNA network, then many of the communications problems are fairly readily resolved provided only that the disruption did not incapacitate a significantly large segment of the communications network.

It is imperative that the back-up plan provide adequate means for back-up of essential communications facilities. They are too frequently neglected in our contingency plans. Each year more data processing time is lost to catastrophically damaged cables, both copper and glass, than is lost to physical damage to all other DP components.

Split Site Management.

It is almost uniformly true that success in bringing any good contingency plan to fruition is dependent upon the support of the director of data processing, by whatever title. In many companies the person in that position has fought long and hard to preserve the integrity of his fiefdom and is, quite understandably, very reluctant to see it fractionated. If the company now has but a single site under each of one or more such persons, it must be anticipated that they might well oppose the establishment of a split-site arrangement unless it is quite clear that they will retain responsibility for both sites.

This frequently encountered opposition by the DP manager to a second site unless it is also under his management is a good thing - sometimes for the wrong reason, but it is usually a good thing. It is difficult to imagine a situation in which both sites of a split-site arrangement should not be under the same management. It is important that control over the two converge at a level not too high for the common management to be fully aware of any activities at either site which might threaten the ability of each to pick up the critical applications.

It is far more likely that two sites will remain compatible if they are under common management than if they are not.

There are, unfortunately, many examples of back-up arrangements within the same organizations, many of which were fully and successfully tested in the past, but where the systems, which were supposed to be and thought by the senior management to be mutually supportive, grew in ways which negated the capability to back up each other.

In a number of the more notable examples, these differences were intentionally introduced by the DP directors to achieve for their particular facility some superior capability or service level not possessed by the other.

Site Locations.

It may not be possible to satisfy all of the desiderata appropriate to the location of split sites. The relative importance of each, and, thus, the need to satisfy it, is best judged in the light of the particular operating environment. The more important ones are these:

1. Proximity.- The two sites should be sufficiently close that it is logistically feasible for each to store the back-up data for the other. In general, this is to say that the two should be within a few hours drive by motor vehicle.

2. Physical Dispersion.- They should be far enough apart that they are not subject to the same causes of disruption provided that the probability of encountering those problems is weighed realistically.

3. Independence.- So far as possible, the facilities should be located so as to be free of dependence upon elements in the infrastructure which are known to be shaky. These include factors ranging from power, communications, water, sewer, transportation, susceptibility to tornadoes and hurricanes, flood plain problems, riot-prone neighborhoods, proximity to major highways and railroads which offer the potential for chemical spills which will require evacuation of the facilities, and other such factors.

THE FUTURE

Distributed processing, the growth of departmental computers, and the rapid proliferation of microcomputers will all contribute in large measure to the problems of contingency planning just as they contribute greatly to both the efficacy and the complexity of our information systems. We can find no reason for an assumption that they will serve to decrease the size of single-site data aggregations to which access will be required for the efficient conduct of our businesses.

The cost of data storage continues to decrease as do access times and both of which serve to accelerate growth in the volume of data to which we want access. It is inevitable that continued growth in data aggregation size will greatly change the nature of contingency plans which are practicable in support of very large, high data-volume business systems.

We expect to see the advent of super-safe, underground DASD facilities connectable to geographically-remote large and small processors through very high-speed communications facilities leaving us with the need to back up only the processors and provide alternate means of communications.

At this time, we find very little reflection in workable contingency plans of recognition of our growing dependence on microcomputers and minis. It may well be that we will not see significant change in that until some major organization has a very serious problem as a result of being unprepared, but we have almost no confidence in that as a motivator of others. The problems of others has not been a primary source of motivation for such contingency planning as we have seen about mainframe facilities. Most such losses are not broadly publicized.

The slowly increasing competence of internal auditors in the technical aspects of data processing should serve, in the foreseeable future, to alert corporate managements to the need for better contingency planning. We expect that, rather than the grief of others, to accelerate the emphasis on contingency plans which address the whole of the data processing dependency as well as recognition that there are many other parts of our businesses other than data processing which, if disrupted, have the potential for causing great harm.

SUMMARY

A wide variety of approaches to contingency planning is available to the persons charged with designing such plans. The task requires innovation, much common sense, rejection of all cookbook approaches, and, above all, prior identification and quantification of the losses potentially averted by the proper plans. No contingency plans are so inherently desirable that they should be implemented without solid economic justification.

Contingency planning is as much dependent upon understanding human nature as it is on understanding the technical aspects of our systems. Unless people at each organizational level from which we need support, or, at least, lack of opposition, can be motivated to support back up and recovery, it is very difficult to put in place. Strong senior management support born of awareness of the need for it contributes more than any other factor to the success of contingency planning - but even that is not a guarantor.

One thing of which we are certain, because it has been demonstrated repeatedly, is that good, workable contingency plans are economically feasible.